IMPLEMENTASI ALGORITME SPECK PADA PROSES PENGIRIMAN *FILE* MULTIMEDIA MELALUI JARINGAN INTERNET

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
David Christanto
NIM: 145150200111189

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

IMPLEMENTASI ALGORITME SPECK PADA PROSES PENGIRIMAN *FILE* MULTIMEDIA MELALUI JARINGAN INTERNET

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
David Christanto
NIM: 145150200111189

Skripsi ini telah diuji dan dinyatakan lulus pada
3 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Ari Kusyanti, S.T., M.Sc.	Mahendra Data, S.Kom., M.Kom
NIP: 19831228 201803 2 002	NIK: 201503 861117 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 3 Agustus 2018

David Christanto

NIM: 145150200111189

KATA PENGANTAR

Segala puji dan syukur kepada Tuhan Yang Maha Esa, atas limpahan Rahmat dan Karunia-Nya, sehingga saya selaku penulis dapat menyelesaikan skripsi dengan judul "IMPLEMENTASI ALGORITME SPECK PADA PROSES PENGIRIMAN *FILE* MULTIMEDIA MELALUI JARINGAN INTERNET" sebagai syarat untuk menyelesaikan Program Sarjana (S1) dan memperoleh gelar Sarjana Komputer (S. Kom.) pada program Sarjana Fakultas Ilmu Komputer Universitas Brawijaya.

Dalam proses penyusunan skripsi ini, saya mengalami berbagai kesulitan. Namun semua kesulitan itu dapat saya lalui berkat adanya dukungan serta bimbingan dari berbagai pihak secara moral dan spiritual. Oleh karena itu, pada kesempatan ini saya selaku penulis menyampaikan ucapan terima kasih kepada:

1. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D selaku dekan Fakultas Ilmu Komputer Universitas Brawijaya.
2. Ibu Ari Kusyanti, S.T, M.Sc. selaku dosen pembimbing I yang telah membimbing dan memberikan dukungan kepada penulis dalam menyusun skripsi ini.
3. Bapak Mahendra Data, S.Kom., M.Kom selaku dosen pembimbing II yang telah membimbing dan memberikan dukungan kepada penulis dalam menyusun skripsi ini.
4. Kedua orang tua saya, Sugianto dan Karima, yang telah memberikan dukungan dan doa yang tiada habisnya kepada saya sehingga saya dapat menyelesaikan skripsi ini.
5. Kakak saya, Eric dan adik saya, Challista, yang memberikan motivasi dan dukungan dalam menyelesaikan skripsi ini.

Akhir kata penulis menyadari bahwa dalam penulisan skripsi ini masih jauh dari kesempurnaan. Karena itu, penulis memohon saran dan kritik yang sifatnya membangun demi kesempurnaannya dan semoga bermanfaat bagi kita semua.

Malang, 7 Juli 2018

Penulis

davidchristanto418@gmail.com

ABSTRAK

Rekaman audio adalah salah satu *file* multimedia yang sering kita jumpai saat ini. Rekaman ini berfungsi untuk menyimpan suara tertentu untuk kebutuhan manusia. Salah satu contohnya adalah rekaman suara dapat menjadi bukti dalam sidang hukum. Alat bukti rekaman suara didukung oleh Pasal 5 ayat 1 UU Nomor 11 Tahun 2008 tentang informasi dan transaksi elektronik yang menyatakan bahwa : "Informasi elektronik dan/atau dokumen elektronik dan/atau hasil cetakannya merupakan alat bukti hukum yang sah". Oleh karena itu, rekaman suara sangat penting untuk dirahasiakan dari pihak yang tidak berwenang. Penelitian ini berfokus pada aspek kerahasiaan. Untuk mendapatkan aspek rahasia, penelitian ini menggunakan algoritme SPECK untuk mengenkripsi rekaman audio. Penulis menggunakan 2 VM PC dalam penelitian ini, yaitu server dan klien. Server dan klien akan melakukan proses DHKE untuk mendapatkan *shared key* yang sama. Kemudian *shared key* digunakan oleh klien untuk mengenkripsi rekaman audio dengan algoritme SPECK. Klien akan mengirimkan *ciphertext* ke server. Terakhir, server akan melakukan dekripsi *ciphertext* agar dapat menjadi rekaman audio. Pengujian yang dilakukan adalah pengujian keamanan, pengujian waktu enkripsi & dekripsi, dan pengujian waktu sistem. Pada pengujian keamanan, hasil *ciphertext* algoritme SPECK sangat berbeda dengan plaintext, sehingga rekaman audio tersebut tidak dapat didengarkan oleh orang ketiga secara langsung. Pengujian waktu enkripsi & dekripsi dilakukan dengan menggunakan ekstensi *file* dan durasi rekaman audio yang berbeda-beda. Ekstensi *file* yang digunakan adalah AIFF, AMR, FLAC, M4A, MP3, OGG, dan WAV. Rata-rata waktu eksekusi enkripsi lebih besar daripada dekripsi. Selisih waktu eksekusi enkripsi dan dekripsi kurang dari 23 detik. Pada pengujian waktu sistem, ekstensi file tidak mempengaruhi lama waktu eksekusi. Waktu eksekusi enkripsi dan dekripsi dipengaruhi oleh ukuran *file* rekaman audio. Semakin besar ukuran *file*, maka semakin lama waktu yang dibutuhkan sistem untuk melakukan enkripsi dan dekripsi.

Kata kunci: rekaman audio, pertukaran kunci publik, enkripsi, DHKE, SPECK.

ABSTRACT

Audio recording is one of the multimedia files that we often encounter today. These recordings serve to store certain sounds for human needs. One example is the sound recording can be evidence in the trial of the law. The proof of sound recording is supported by Article 5 paragraph 1 of Law Number 11 Year 2008 regarding information and electronic transactions stating that: "Electronic information and / or electronic documents and / or prints are legal legal evidence". Therefore, sound recording is very important to be kept secret from unauthorized parties. This study focuses on the aspect of secrecy. To get the secret aspect, this research uses SPECK algorithm to encrypt audio recordings. The author uses 2 VM PCs in this research, ie server and client. The server and client will perform the DHKE process to get the same shared key. Then the shared key is used by the client to encrypt audio recordings with SPECK algorithm. The client will send the ciphertext to the server. Finally, the server will decrypt the ciphertext to be an audio recording. Tests conducted are security testing, testing time of encryption & decryption, and system time test. In security testing, the SPECK algorithm's ciphertext results are very different from plaintext, so the audio recording can not be heard by a third person directly. Time encryption & decryption testing is done using different file extensions and audio recording durations. The file extensions used are AIFF, AMR, FLAC, M4A, MP3, OGG, and WAV. The average execution time of encryption is greater than decryption. Difference in execution time for encryption and decryption is less than 23 seconds. In system time testing, file extensions do not affect the execution time. The execution time of encryption and decryption is affected by the size of the audio recording file. The larger the file size, the longer it takes the system to perform encryption and decryption.

Keywords: *audio recording, public key exchange, encryption, DHKE, SPECK.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Rekaman audio merupakan salah satu *file* multimedia yang sering kita jumpai saat ini. Rekaman ini berfungsi untuk menyimpan suara tertentu untuk kebutuhan manusia. Salah satu contohnya adalah rekaman suara dapat menjadi bukti dalam sidang hukum. Alat bukti rekaman suara didukung oleh Pasal 5 ayat 1 UU Nomor 11 Tahun 2008 tentang informasi dan transaksi elektronik yang menyatakan bahwa : "Informasi elektronik dan/atau dokumen elektronik dan/atau hasil cetakannya merupakan alat bukti hukum yang sah". Pasal 5 ayat (1) UU ITE dapat dikelompokkan menjadi dua bagian. Pertama, informasi elektronik dan/atau dokumen elektronik. Kedua, hasil cetak dari informasi elektronik dan/atau hasil cetak dari dokumen elektronik (Sitompul, 2012). Informasi elektronik dan dokumen elektronik tersebut akan menjadi alat bukti elektronik (*Digital Evidence*). Sedangkan hasil cetak dari informasi elektronik dan dokumen elektronik akan menjadi alat bukti surat.

Pada Tahun 2015, Purwosasongko melakukan penelitian untuk mengukur kekuatan bukti rekaman suara dalam proses pemberantasan tindak pidana korupsi. Penelitian ini menggunakan metode pendekatan yuridis empiris yang melakukan pendekatan secara langsung terhadap masalah baik dari perpekstif perundang-undangan maupun praktik langsung terhadap penegakan hokum yang ada di Komisi Pemberantasan Korupsi Jakarta dengan cara wawancara dengan beberapa staf KPK terkait. Hasil yang didapatkan menunjukkan bahwa alat bukti rekaman suara mempunyai kedudukan dan kekuatan yang sama dengan alat bukti lain dalam lingkup mengenai proses penyidikannya. Sebagai lembaga peradilan yang independen, KPK telah berhasil menangani berbagai perkara korupsi yang di lakukan oleh para lapisan masyarakat baik para pejabat tinggi ataupun oleh para penegak hukum itu sendiri, dimana hanya lembaga KPK yang menggunakan alat bukti rekaman suara dalam menggungkap kasus korupsi di Indonesia dan hanya lembaga peradilan KPK yang memiliki kewenangan tersebut (Purwosasongko, 2015). Oleh karena itu, proses pengiriman rekaman suara sangat penting untuk dirahasiakan dari pihak publik di internet.

Ada beberapa penelitian tentang keamanan rekaman audio, yaitu pertama, enkripsi audio dengan menggunakan algoritme LOKI97 dan yang kedua, enkripsi audio dengan menggunakan algoritme *Transposition Cipher*, serta ketiga adalah penelitian dari Ray Beaulieu dan teman-teman tentang algoritme SPECK. Pada penelitian menggunakan algoritme LOKI97, hasil uji didapatkan dari perbandingan nilai *Root Mean Square*(RMS). Rata-rata hasil uji RMS antara *file* audio sebelum dienkripsi dengan *file* audio setelah dienkripsi adalah 108,0052. Sedangkan rata-rata hasil uji RMS antara *file* audio sebelum dienkripsi dengan *file* audio setelah dienkripsi adalah 0(berarti tidak ada perbedaan). Dari hasil ini dapat disimpulkan, LOKI97 dapat mengamankan audio dengan baik(Adhika, 2012). Sedangkan pada penelitian menggunakan *transposition cipher,* hasil diuji

dengan menggunakan *Mean Square Error*(MSE). Pengujian ini dilakukan dengan menggunakan kunci yang benar dan salah. Pada kasus menggunakan kunci benar, *file* enkripsi mendapatkan nilai 0,00000362, sedangkan pada *file* dekripsi mendapatkan nilai 0. Hal ini menandakan bahwa hasil dari enkripsi dan dekripsi berhasil. Pada kasus menggunakan kunci salah, *file* enkripsi mendapatkan nilai 0,00000362 dan *file* dekripsi mendapatkan nilai 0,00000428. *File* dekripsi tidak kembali menjadi *file* aslinya karena menggunakan kunci yang salah. Hal ini membuktikan bahwa Transposition Cipher dapat mengamankan *file* audio (Jawahir & Haviluddin, 2015). Penelitian selanjutnya dilakukan oleh Ray Beaulieu berjudul "The Simon and SPECK Families of Lightweight Block Ciphers". Algoritme SPECK telah dirilis oleh *National Security Agency* (NSA). Algoritme SPECK telah diuji sendiri oleh kripto analis dari NSA dengan hasil tidak ada celah yang ditemukan (Beaulieu, R., *et al*, 2013).

Walau algoritme LOKI97 dan *transposition cipher* dapat mengamankan audio dengan baik, tetapi kedua algoritme tersebut sudah memiliki banyak celah. Hal ini mengakibatkan algoritme LOKI97 dan *transposition cipher* tidak disarankan untuk digunakan sebagai algoritme untuk merahasiakan data. Oleh karena kelemahan tersebut, penulis ingin mengembangkan penelitian tersebut dengan mengangkat judul skripsi "IMPLEMENTASI ALGORITME SPECK PADA PROSES PENGIRIMAN *FILE* MULTIMEDIA MELALUI JARINGAN INTERNET".

1.2 Rumusan masalah

Berdasarkan latar belakang yang ada, maka dapat dirumuskan masalah sebagai berikut :

1. Bagaimana menerapkan algoritme SPECK yang dapat memberikan aspek kerahasiaan terhadap rekaman audio?
2. Berapa waktu eksekusi yang diperlukan untuk melakukan enkripsi dan dekripsi dengan algoritme SPECK?
3. Bagaimanakah perbandingan waktu eksekusi algoritme SPECK dengan variasi ekstensi *file* pada rekaman audio?

1.3 Tujuan

Tujuan dari penelitian ini untuk menjawab rumusan masalah, yaitu :

1. Mengetahui penerapan algoritme SPECK dalam memberikan aspek kerahasiaan terhadap rekaman audio.
2. Mengetahui waktu eksekusi yang diperlukan untuk melakukan enkripsi dan dekripsi dengan algoritme SPECK.
3. Mengetahui perbandingan waktu eksekusi algoritme SPECK dengan variasi ekstensi *file* pada rekaman audio.

1.4 Manfaat

Adapun manfaat dari penelitian ini adalah:

1. Pengguna akan mendapatkan aspek kerahasiaan saat mengirim rekaman audio dengan cara mengenkripsi data rekaman audio.
2. Pengguna tidak perlu untuk khawatir dengan kerahasiaan rekaman audio ini, karena telah dienkripsi dengan algoritme SPECK yang telah diuji oleh NSA pada Juni 2013.

1.5 Batasan masalah

Dalam penelitian ini terdapat batasan-batasan, sebagai berikut :

1. *File* multimedia hanya dibatasi pada *file* rekaman audio.
2. Penelitian ini tidak membahas tampilan sistem.
3. Penelitian ini berfokus pada aspek kerahasiaan.
4. Metode pertukaran *secret key* menggunakan *Diffie-Hellman Key Exchange*
5. Pada Algoritme SPECK, besar blok (2n) menggunakan 128 bit dan besar kunci 128 bit
6. *User* memasukan kunci dan *file* yang akan dikirim ke dalam sistem secara manual.

1.6 Sistematika pembahasan

Sistematika penulisan memberikan uraian dari penyusunan skripsi secara garis besar yang meliputi, antara lain :

BAB I PENDAHULUAN

Memuat tentang latar belakang penelitian tentang keamanan pada rekaman audio dengan algoritme SPECK, rumusan masalah, tujuan, manfaat, batasan masalah, dan sistematika penulisan.

BAB II LANDASAN KEPUSTAKAAN

Menjelaskan kajian pustaka berupa algoritme SPECK dan penelitian terdahulu dan dasar teori / referensi tentang audio, kriptografi, algoritme SPECK, dan *Diffie-Hellman Key Exchange*.

BAB III METODOLOGI

Menguraikan langkah-langkah yang digunakan dalam melakukan penelitian mengenai sistem keamanan terhadap pengiriman rekaman audio.

BAB IV ANALISIS KEBUTUHAN DAN PERANCANGAN

Menjelaskan analisis dan perancangan sistem keamanan rekaman audio yang menggunakan algoritme SPECK. Hal ini dilakukan agar dapat menjawab permasalahan yang telah dijabarkan di rumusan masalah.

BAB V IMPLEMENTASI

Memuat implementasi sistem keamanan dengan menerapkan algoritme SPECK. Objek dari sistem ini adalah rekaman audio.

BAB VI PENGUJIAN DAN ANALISIS

Melakukan pengujian *test vector*, pengujian fungsional, pengujian validitas enkripsi dan dekripsi, pengujian waktu enkripsi dan dekripsi, pengujian waktu sistem, dan pengujian keamanan. Hal ini dilakukan untuk mendapatkan jawaban dari rumusan masalah.

BAB VII PENUTUP

Memuat kesimpulan yang diperoleh dari hasil pengujian sistem tentang pengamanan rekaman audio. Serta terdapat saran yang dapat dipertimbangkan untuk penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini menjabarkan tentang kajian pustaka penelitian sebelumnya, audio, kriptografi, algoritme SPECK, dan *Diffie-Hellman Key Exchange* (DHKE).

2.1 Kajian Pustaka

Ray Beaulieu (2013) mengamati banyak algoritme *block cipher* yang disimpan pada kapasitas kecil masih memiliki kelemahan. Algoritme-algoritme tersebut dapat berjalan bagus pada satu platform, tetapi tidak menyediakan performa tinggi pada area alat tersebut. Oleh karena itu, Ray mengusulkan 2 algoritme *block cipher* baru, yaitu SIMON dan SPECK untuk menutup kelemahan tersebut. Algoritme SIMON memiliki performa yang optimal di hardware, sedangkan algoritme SPECK memiliki performa optimal di software. Kedua algoritme ini memiliki berbagai macam ukuran blok dan kunci.

Adhika (2012) melakukan penelitian tentang enkripsi dan dekripsi audio khususnya dengan format AMR dengan algoritme LOKI97. Loki97 adalah algoritme *block cipher* yang menggunakan 128-bit data dan 128/192/256 bit kunci (Lawrie Brown, 1998). Algoritme ini termasuk kandidat *Advanced Encryption Standard* (AES) yang diajukan kepada *National Institute of Standards and Technology* (NIST). Untuk memudahkan implementasi algoritme LOKI97, maka Adhika membuat aplikasi dengan bahasa pemrograman Delphi 7.0. Adhika mendapatkan hasil *ciphertext* relatif aman. Hasil ini didapatkan berdasarkan pengujian dengan dilakukan pembandingan nilai *Root Mean Square* (RMS). Rata-rata hasil uji RMS antara *file* audio sebelum dienkripsi dengan *file* audio setelah dienkripsi adalah 108,0052. Sedangkan rata-rata hasil uji RMS antara *file* audio sebelum dienkripsi dengan *file* audio setelah dienkripsi adalah 0 (artinya tidak ada perbedaan).

Ahmad Jawahir dan Haviluddin (2015) mencoba untuk mengimplemtasikan enkripsi audio dengan algoritme *transposition cipher*. Untuk membuktikan hasilnya, Ahmad melakukan pengujian menggunakan *Mean Square Error* (MSE). Bila MSE = 0, maka *file* tersebut sama (tidak ada perubahan) dengan *file* aslinya, bila MSE ≠ 0, maka *file* tersebut berbeda dengan *file* aslinya. Pengujian ini dilakukan dengan menggunakan kunci yang benar dan salah. Pada kasus menggunakan kunci benar, *file* enkripsi mendapatkan nilai 0,00000362, sedangkan pada *file* dekripsi mendapatkan nilai 0. Hal ini menandakan bahwa hasil dari enkripsi dan dekripsi berhasil. Pada kasus mengguanakan kunci salah, *file* enkripsi mendapatkan nilai 0,00000362 dan *file* dekripsi mendapatkan nilai 0,00000428. *File* dekripsi tidak kembali menjadi *file* aslinya karena menggunakan kunci yang salah.

2.2 Audio

Audio adalah gelombang yang memiliki komponen seperti panjang gelombang, amplitudo, dan frekuensi. Komponen tersebut yang menyebabkan audio dapat berbeda. Amplitudo merupakan kekuatan gelombang sinyal. Audio yang memiliki amplitudo yang kecil akan terdengar lebih kecil. Frekuensi adalah jumlah siklus yang terjadi per detik. Getaran gelombang audio yang cepat membuat frekuensi semakin tinggi. Orang menyanyi dengan nada tinggi, maka audio yang akan dihasilkan memiliki frekuensi yang besar pula.

Rekaman audio memiliki *bitrate*, *bit depth*, dan *sample rate*. *Bitrate* dalam bidang multimedia merepresentasikan seberapa besar informasi yang disimpan per satuan waktu rekaman. *Bitrate* dapat dipengaruhi oleh beberapa faktor, yaitu:

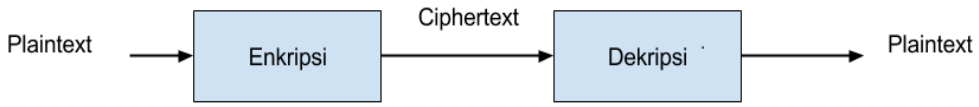
a. Frekuensi
b. Besar bit dalam samples
c. Cara melakukan proses encode
d. Algoritme untuk melakukan kompresi audio

Bit depth adalah banyak bit informasi dalam setiap sample. Contohnya adalah pada CD digital audio dan DVD-audio. CD digital audio menggunakan 16 bit per *sample*, sedangkan pada DVD-audio menggunakan 24 bit per *sample*. *Bit depth* akan mempengaruhi *bit rate*, *level noise*, dan besar *file* audio tersebut.

Sample rate adalah rata-rata *sample* (dalam bentuk frekuensi) yang didapatkan selama 1 detik dari sinyal yang diterima secara terus menerus (*continuous signal*) menjadi sinyal yang terpisah (*discrete signal*). *Sample rate* yang umumnya digunakan adalah 44,1 kHz, 48 kHz, 88,2 kHz, 96 kHz, dan 192 kHz. Standar audio CD menggunakan sample rate 44,1 kHz (44100 Hz). Hal ini dikarenakan telinga manusia hanya dapat menangkap frekuensi suara dari 20Hz sampai 20kHz. Sehingga sample paling cocok adalah 44,1 kHz (*Nyquist-Shannon Sampling Theory*)

2.3 Kriptografi

Pesan merupakan data yang dapat dibaca serta dimengerti maknanya. Pesan dapat disebut juga dengan *plaintext.* Enkripsi merupakan proses untuk menyembunyikan pesan dalam bentuk lain yang susah dimengerti untuk menyembunyikan substansinya (Schneier, 1996). *Ciphertext* merupakan pesan yang telah dilakukan proses enkripsi. Proses untuk mengembalikan *ciphertext* menjadi *plaintext* disebut dengan dekripsi. Keterkaitan tersebut dapat digambarkan pada Gambar 2.1.



Gambar 2.1 Enkripsi dan Dekripsi
Sumber : (Schneier, 1996)

Menurut Stallings (2005), kriptografi adalah bagian dari kriptologi yang berhubungan dengan desain algoritma untuk enkripsi dan dekripsi yang bertujuan untuk metakinkan kerahasiaan dan keaslian pesan.

Konsep penggunaan kriptografi menurut Fidens (2006) yaitu :

1. Kerahasiaan (*Confidentiallity*) yaitu proses penyembunyian data dari orang-orang yang tidak mempunyai otoritas
2. Integritas (*Integrity*) yaitu prises untuk menjaga agar sebuah data tidak diubah-ubah waktu dikirim maupun disimpan.
3. Penghindaran penolakan (*Non-Repudiation*) yaitu proses untuk menjaga bukti-bukti bahwa suatu data berasal dari seseorang.
4. Autentikasi (*Autenthication*) yaitu proses untuk menjamin keaslian data.

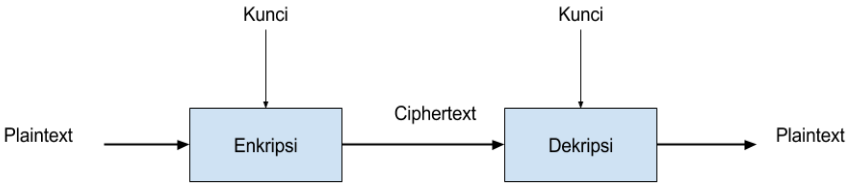
2.3.2 Algoritma Kriptografi

Algoritma kriptografi (*chipper*) adalah fungsi matematika yang digunakan untuk enkripsi dan dekripsi (Schneier, 1996). Berdasarkan kunci yang dipakai untuk proses dekripsi bisa menggunakan kunci yang sama dengan kunci enkripsi (kunci simetris) maupun kunci yang berbeda saat proses enkripsi (kunci asimetris) (Ariyus, 2008).

1) Algoritma kunci simetris (*symmetric-key cryptographic*)

Dalam algoritma kunci simetris, kunci yang digunakan untuk melakukan enkripsi sama dengan kunci yang digunakan untuk dekripsi (Schneier, 1996). Istilah lain yang digunakan untuk algoritma ini adalah algoritma kunci privat. Keamanan kriptografi ini terletak pada kerahasiaan kuncinya. Contoh algoritma kunci simetris adalah DES (*Data Encryption Standard*), *Blowfish*, *Twofish*, *AES, SIMON,* dan *SPECK*.

Proses enkripsi dan dekripsi algoritma kunci simetris dapat dilihat pada Gambar 2.2

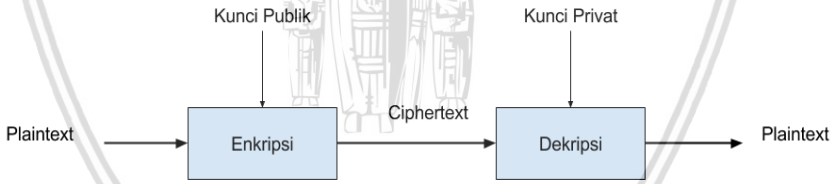


Gambar 2.2 Enkripsi dan Dekripsi Algoritme Kunci Simetris
Sumber : (Schneier, 1996)

Algoritma kriptografi simetris dibagi menjadi dua jenis yaitu algoritma aliran (*stream ciphers*) serta algoritma blok (*block ciphers*). Pada algoritma aliran (*stream ciphers)*, proses enkripsi berorientasi pada satu bit atau satu *byte* data. Sedangkan untuk algoritma blok (*block cipher*), proses enkripsi berorientasi pada sekumpulan bit atau *byte* data (per blok) (Kurniawan, 2004).

2) Algoritma kunci asimetris

Algoritma kunci asimetris merupakan algoritma dengan kunci enkripsi serta dekripsi yang berbeda (Schneier, 1996). Algoritma ini disebut juga algoritma kriptografi kunci publik. Dalam algoritma ini, setiap orang yang memiliki kunci publik dapat melakukan proses enkripsi suatu pesan, data ataupun informasi, namun untuk melakukan proses dekripsi dari *ciphertext* hanya orang yang memiliki kunci privat. Proses enkripsi dan dekripsi algoritma kunci asimetris dapat dilihat pada Gambar 2.3



Gambar 2.3 Enkripsi dan Dekripsi Algoritme Kunci Asimetris
Sumber : (Schneier, 1996)

2.4 Algoritme SPECK

Algoritme SPECK adalah salah satu algoritme *block cipher* yang telah dirilis oleh National Security Agency (NSA) pada Juni 2013. Karena *block cipher*, maka algoritme SPECK menggunakan input (masukan) berupa blok (kumpulan bit) bukan bit. SPECK termasuk aplikasi yang ringan dan sangat baik dijalankan di dalam *software*. SPECK dirancang untuk dapat berjalan baik di *software* maupun *hardware*, terutama di dalam *microcontroller*. Algoritme ini memiliki besar kunci dan blok yang berbeda-beda tergantung pada kasus yang dihadapi. Satu blok akan terdiri 2 word. 1 word dapat berukuran 16, 24, 32, 48, atau 64 bit (Beaulieu, R., *et al*, 2013).

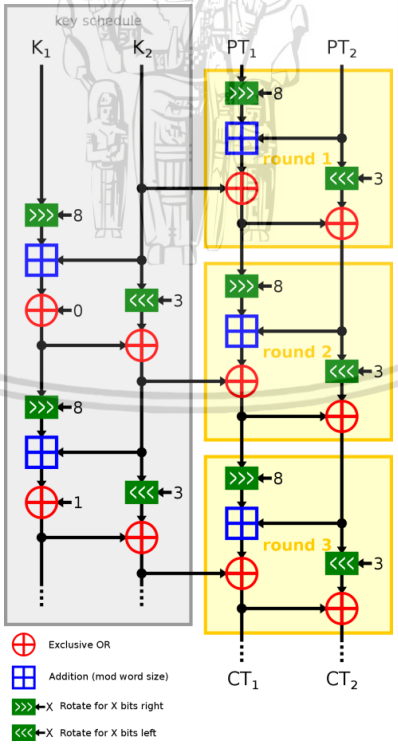
Besar blok dan kunci akan menentukan seberapa banyak ronde yang akan dipakai, rinciannya dapat dilihat pada Tabel 2.1.

Tabel 2.1 Versi Algoritme SPECK

Block Size (bits)	***Key Size (bits)***	***Rotation α***	***Rotation β***	***Rounds***
2 x 16 = 32	4 x 16 = 64	7	2	22
2 x 24 = 48	3 x 24 = 72	8	3	22
	4 x 24 = 96			23
2 x 32 = 64	3 x 32 = 96	8	3	26
	4 x 32 = 128			27
2 x 48 = 96	2 x 48 = 96	8	3	28
	3 x 48 = 144			29
2 x 64 = 128	2 x 64 = 128	8	3	32
	3 x 64 = 192			33
	4 x 64 = 256			34

Sumber : (Ray Beaulieu et al, 2013)

Ilustrasi algoritme SPECK seperti pada Gambar 2.4 berikut.



Gambar 2.4 Ilustrasi SPECK dengan 3 Ronde dan 2 *Word Key*
Sumber : (Wikipedia.org)

Pada Gambar 2.4, SPECK dibagi 2 bagian yaitu fungsi *key schedule* dan fungsi ronde.

2.4.1 Fungsi *Key Schedule*

Key schedule pada algoritme SPECK digunakan untuk membuat kunci setiap ronde. Kunci yang telah dibuat akan digunakan di fungsi ronde setiap ronde. Pada proses ini terdapat 2 variabel yang akan diproses, yaitu k_i an l_i.

Untuk mendapatkan nilai l_i dilakukan rumus sebagai berikut

$$l_{i+m-1} = (k_i + S^{-\alpha} l_i) \oplus \mathrm{i} \tag{2.1}$$

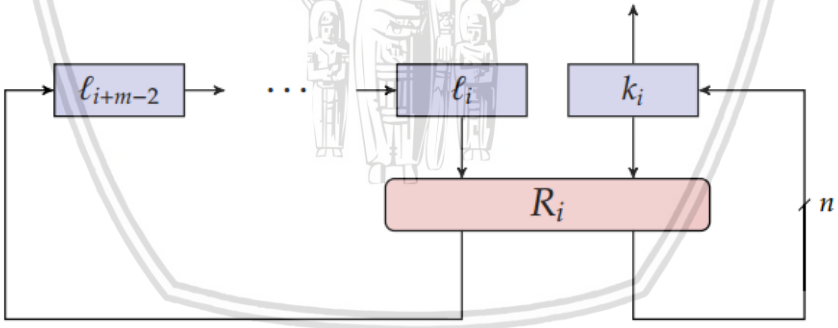
Pada rumus 2.1, nilai m adalah nilai *word* pada *key* (tergantung jenis SPECK yang digunakan). Nilai i adalah ronde saat ini. Nilai i akan terus bertambah dari 0 sampai jumlah ronde sesuai jenis SPECK yang digunakan. Nilai l_{i+m-1} didapatkan dengan cara melakukan *addition modulo* 2^n antara k_i dengan l_i yang telah dilakukan rotasi ke kanan sejumlah α bit. Kemudian hasilnya akan dilakukan operasi XOR dengan nilai i.

Untuk mendapatkan nilai k_i, dilakukan rumus sebagai berikut

$$k_{i+1} = S^{\beta} k_i \oplus l_{i+m-1} \tag{2.2}$$

Pada rumus 2.2, nilai k_{i+1} didapatkan dengan cara melakukan operasi XOR antara k_i yang telah dilakukan rotasi ke kiri sejumlah β bit dengan nilai l_{i+m-1} yang didapat dari rumus 2.1.

Kedua rumus ini akan dilakukan perulangan jumlah ronde dari jenis SPECK yang digunakan. Ilustrasi gambaran proses *key schedule* pada algoritme SPECK dapat dilihat pada Gambar 2.5.



Gambar 2.5 Ilustrasi *Key Schedule* pada Algoritme SPECK
Sumber : (Ray Beaulieu et al, 2013)

2.4.2 Fungsi *Round*

Operasi yang dilakukan dalam fungsi ronde ada 3, yaitu operasi *addition modulo* 2^n, rotasi ke kiri atau ke kanan sejumlah bit tertentu, dan XOR. Operasi *addition modulo* 2^n sama seperti melakukan operasi AND atau pertambahan per bit (Beaulieu, R., *et al*, 2013). Pada fungsi ronde ini terdapat rumus enkripsi dan dekripsi yang dipakai pada algoritme SPECK.

Proses enkripsi dilakukan dengan rumus berikut

$$R_k(x, y) = ((S^{-\alpha} x + y) \oplus \mathrm{k} \ , \ S^{\beta} y \oplus ((S^{-\alpha} \mathrm{x} + \mathrm{y}) \oplus \mathrm{k})) \tag{2.3}$$

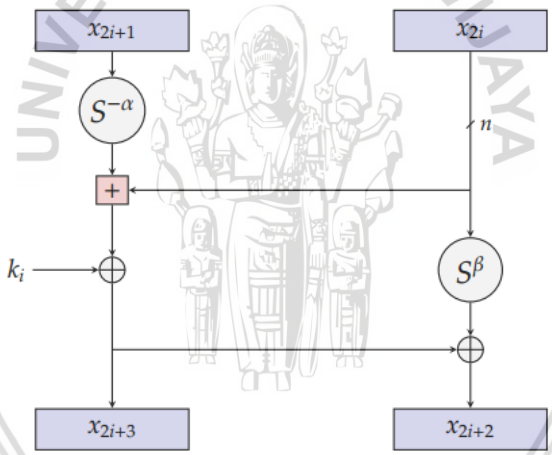
Pada rumus 2.3, x adalah *plaintext* pertama dan y adalah *plaintext* kedua. X didapatkan dari melakukan *addition modulo* 2^n antara hasil x yang dilakukan rotasi ke kanan sejumlah α bit dan y. Kemudian hasil tersebut dilakukan operasi XOR terhadap *k* (key). Sedangkan Y didapatkan dari melakukan operasi rotasi ke kiri sejumlah β bit pada y. Kemudian hasil tersebut dilakukan operasi XOR terhadap x yang telah dimasukan rumus 2.3.

Proses dekripsi dilakukan dengan rumus berikut

$$R_k{}^{-1}(x,y) = (S^{\alpha}((x \oplus \mathrm{k}) - S^{-\beta}(x \oplus y)) \ , S^{-\beta}(\mathrm{x} \oplus \mathrm{y})) \qquad (2.4)$$

Pada rumus 2.4, x didapatkan dengan melakukan operasi XOR pada x dengan *k* (key). Hasil tersebut akan dilakukan rotasi ke kiri sejumalah α bit. Kemudian akan dikurangkan dengan hasil dari y yang sudah diproses pada rumus 2.4.Sedangkan y didapatkan dari melakukan operasi XOR antara x dan y. Kemudian hasilnya akan dirotasi ke kanan sejumlah β bit.

Kedua proses ini akan diulang sejumlah ronde sesuai pada jenis SPECK yang digunakan. Ilustrasi gambaran fungsi ronde pada SPECK dapat dilihat pada Gambar 2.6.



Gambar 2.6 Ilustrasi fungsi Ronde pada Algoritme SPECK
Sumber : (Ray Beaulieu et al, 2013)

2.5 *Diffie-Hellman Key Exchange* (DHKE)

Diffie-Hellman key exchange adalah suatu metode tukar kunci kriptografi dengan aman di publik. DHKE menggunakan konsep awal dari Ralph Merkle kemudian dipublikasikan oleh Whitfield Diffie dan Martin Hellman pada tahun 1976.

DHKE menggunakan beberapa parameter umum, yaitu sebagai berikut:

a. Terdapat 2 atau lebih orang *user* yang berinteraksi

b. Terdapat kesepakatan bilangan prima, p dan g sedemikian sehingga g < p

c. Bilangan prima p dan g tidak dirahasiakan, sehingga antar *user* dapat bertukar kunci di lingkungan yang tidak aman sekalipun

Ilustrasi proses dalam pertukaran kunci menggunakan algoritme DHKE dengan 2 orang *user*, sebagai berikut:

1. Kedua *user* melakukan perjanjian angka untuk nilai p dan g
2. *User* pertama memilih bilangan secara acak sebagai *secret key* (X_a), menghitung nilai (Y_a) dan mengirimkan pada *user* kedua. Lihat Persamaan 2.5

$$Y_a = g^{X_a} \, mod \, p \tag{2.5}$$

3. *User* kedua memilih bilangan secara acak sebagai *secret key* (X_b), menghitung nilai (Y_b) dan mengirimkan pada *user* pertama. Lihat Persamaan 2.6

$$Y_b = g^{X_b} \, mod \, p \tag{2.6}$$

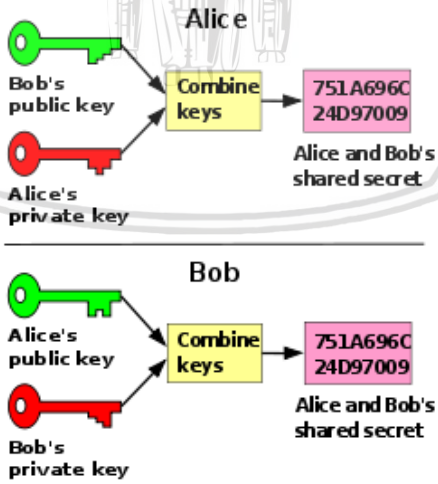
4. *User* pertama menghitung *shared key* (S_a). Lihat Persamaan 2.7

$$S_a = {Y_b}^{X_a} \, mod \, p \tag{2.7}$$

5. *User* kedua menghitung *shared key* (S_b). Lihat Persamaan 2.8

$$S_b = {Y_a}^{X_b} \, mod \, p \tag{2.8}$$

6. *Shared key* pada *user* pertama(S_a) dan *user* kedua(S_b) memiliki nilai yang sama.



Gambar 2.7 Mekanisme *Diffie-Hellman Key Exchange*
Sumber : (Schneier, 1996)

BAB 3 METODOLOGI

Bab ini berisi langkah-langkah yang digunakan dalam melakukan penelitian ini. Tahapan pengerjaan penelitian ini dapat dilihat pada Gambar 3.1.

Gambar 3.1 Metodologi Penelitian

3.1 Studi Literatur

Literatur yang digunakan untuk menunjang penelitian tentang implementasi Algoritme SPECK pada rekaman audio terdapat pada sub-bab studi literatur ini. Dasar teori untuk mendukung penulisan skripsi didapat dari buku, jurnal, situs web resmi, dan penelitian sebelumnya dengan pembahasan yang sama atau

berhubungan dengan skripsi. Pokok bahasan dalam penelitian ini adalah cara untuk mengimplementasikan algoritme SPECK pada rekaman audio.

3.2 Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk mengidentifikasi kebutuhan dari sistem enkripsi rekaman audio dengan algoritme SPECK. Pada algoritme SPECK dilakukan analisis kebutuhan terkait dengan besar kunci dan besar blok sebagai input dari algorime SPECK.

3.3 Perancangan

Perancangan dilakukan setelah mendapatkan hasil dari analisis kebutuhan. Pada penelitian ini dibutuhkan perancangan algoritme, sistem, dan pengujian. Perancangan algoritme dilakukan agar program penulis sesuai dengan algoritme SPECK di jurnal Ray Beaulieu. Perancangan sistem keamanan dilakukan agar menggambarkan cara kerja sistem dalam mengamankan rekaman data melalui proses enkripsi dengan algoritme SPECK. Perancangan pengujian berfungsi untuk mengecek sistem telah berjalan dengan benar atau belum.

3.4 Implementasi

Pada tahap ini, penulis akan merealisasikan perancangan pada tahap sebelumnya. Proses implementasi dimulai dari membangun spesifikasi lingkungan sistem yang telah dirancang sebelumnya. Spesifikasi terdiri dari spesifikasi hardware dan software yang dibutuhkan oleh sistem. Kemudian penulis membuat sistem enkripsi dan dekripsi terhadap rekaman audio dengan menggunakan algoritme SPECK.

3.5 Pengujian dan Analisis

Pengujian sistem dilakukan agar mengetahui apakah sistem telah berjalan secara benar atau belum. Berikut beberapa pengujian yang akan dilakukan:

1. Pengujian Test Vector

   Pengujian ini berfungsi untuk memastikan algoritme SPECK penulis sesuai dengan algoritme SPECK di jurnal "*The Simon and SPECK Families of Lightweight Block CIphers*"

2. Pengujian Fungsional

   Pengujian ini bertujuan untuk memastikan semua fitur dalam sistem dapat berjalan sesuai fungsinya.

3. Pengujian Validitas Enkripsi dan Dekripsi

   Pengujian ini bertujuan untuk memastikan plainteks sama dengan hasil dekripsinya. Pengujian ini akan menggunakan *input* rekaman audio dengan variasi ekstensi *file* dan durasi.

4. Pengujian Waktu Enkripsi dan Dekripsi

   Pengujian ini berfungsi untuk mengetahui waktu enkripsi dan dekripsi dari algoritme SPECK. Pengujian ini juga menggunakan *input* rekaman audio dengan variasi ekstensi *file* dan durasi.

5. Pengujian Waktu Sistem

   Pengujian ini bertujuan untuk membandingkan waktu eksekusi algoritme SPECK dengan variasi ekstensi *file* pada rekaman audio.

6. Pengujian Keamanan

   Pengujian ini berfungsi untuk mengetahui tingkat keamanan bila diserang *man in the middle*.

3.6 Kesimpulan dan Saran

Kesimpulan mengenai keamanan rekaman audio akan diambil dari hasil pengujian dan analisis penelitian. Kemudian, saran akan ditujukan kepada penelitian selanjutnya agar penelitian selanjutnya dapat menyempurnakan penelitian ini.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN

Bab ini berisi kebutuhan-kebutuhan apa saja yang akan digunakan dalam penelitian beserta rancangan penelitian ini baik rancangan algoritme, sistem dan pengujian yang akan dilakukan.

4.1 Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk mengidentifikasi kebutuhan dari sistem keamanan pada proses pengiriman rekaman audio dengan algoritme SPECK.

4.1.1 Kebutuhan Fungsional

Pada penelitian ini, terdapat kebutuhan fungsional, yakni:

1. Klien harus dapat membuka lokasi rekaman audio yang akan dikirim.
2. Klien dan server harus mampu menerima inputan *public key* berupa huruf maupun angka.
3. Klien dan server dapat bertukar data berupa *public key* untuk diproses menjadi *shared key* dengan DHKE.
4. Klien harus bisa melakukan enkripsi dengan algoritme SPECK.
5. Klien harus dapat mengirimkan *ciphertext* ke Server.
6. Server harus mampu melakukan dekripsi dengan algoritme SPECK.
7. Server harus bisa menyimpan rekaman audio setelah didekripsi di lokasi tertentu.

4.1.2 Kebutuhan Perangkat Keras

Dalam penelitian ini membutuhkan perangkat keras, sebagai berikut:

1. Laptop, dengan spesifikasi sebagai berikut:
   - Model : Lenovo G40-70
   - OS : Windows 10 Enterprise (64 bit)
   - Processor : Intel (R) Core (TM) i5-5200U CPU (4 CPUs) @ 2.20GHz
   - RAM : 8 GB
   - Fungsi : Untuk menjalankan *Virtual Machine* Klien dan Server
2. *Virtual Machine* Klien dan Server, dengan spesifikasi sebagai berikut:
   - OS : Windows XP (32 bit)
   - Processor : Intel (R) Core (TM) i5-5200U CPU (1 CPUs) @ 2.20GHz
   - HDD : 5 GB
   - RAM : 1 GB

- Fungsi : Untuk menjalankan sistem keamanan pada pengiriman rekaman audio

4.1.3 Kebutuhan Perangkat Lunak

Dalam pengerjaan penelitian ini, penulis membutuhkan perangkat lunak sebagai berikut:

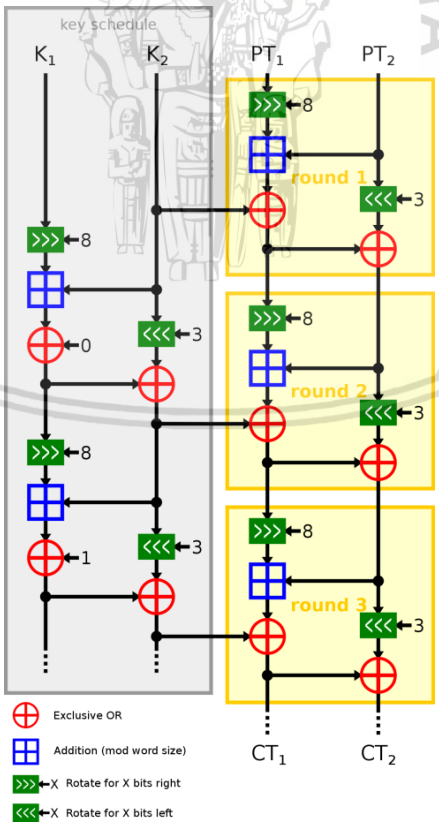
1. Java (dengan pengaturan java heap space berukuran 1024 MB)
2. NetBeans IDE
3. Media Player Classic

4.2 Perancangan

Pada sub bab ini menjelaskan perancangan baik algoritme, sistem, dan pengujian untuk membantu penulis dalam melakukan implementasi sistem.

4.2.1 Perancangan Algoritme

Algoritme SPECK dibuat oleh Ray Beaulieu dan teman-temannya untuk mengatasi kelemahan-kelemahan algoritme *block cipher* sebelumnya. Pada SPECK terdapat fungsi untuk membuat kunci (*Key Schedule*) dan fungsi yang dilakukan setiap ronde. Rancangan algoritme dapat dilihat pada Gambar 4.1



Gambar 4.1 Rancangan SPECK 128/128

Rumus yang akan digunakan dalam fungsi key schedule dapat dilihat pada Rumus 4.1 dan Rumus 4.2.

$$l_{i+m-1} = (k_i + S^{-\alpha} l_i) \oplus \mathrm{i} \tag{4.1}$$

$$k_{i+1} = S^{\beta} k_i \oplus l_{i+m-1} \tag{4.2}$$

Persamaan enkripsi dapat dilihat pada Rumus 4.3

$$R_k(x,y) = ((S^{-\alpha}x + y) \oplus \mathrm{k}\ ,\ S^{\beta}y \oplus ((S^{-\alpha}\mathrm{x} + \mathrm{y}) \oplus \mathrm{k})) \tag{4.3}$$

Persamaan dekripsi dilakukan dengan Rumus 4.4

$$R_k{}^{-1}(x,y) = (S^{\alpha}((x \oplus \mathrm{k}) - S^{-\beta}(x \oplus y))\ ,\ S^{-\beta}(\mathrm{x} \oplus \mathrm{y})) \tag{4.4}$$

Rumus 4.1 - 4.4 berdasarkan landasan kepustakaan tentang algoritme SPECK pada sub bab 2.4. Di dalam jurnal Ray Beaulieu, terdapat *plaintext*, *key*, beserta *ciphertext* sebagai *text vector* pada SPECK yang dapat dilihat pada Tabel 4.1.

Tabel 4.1 *Test Vector* Algoritme SPECK 128 / 128

SPECK 128 / 128		
Key	0f0e0d0c0b0a0908	0706050403020100
Plaintext	6c61766975716520	7469206564616d20
Ciphertext	a65d985179783265	7860fedf5c570d18

Sumber : (Ray Beaulieu et al, 2013)

Pada penelitian ini, rekaman audio memiliki ukuran bit yang besar dan kunci yang dibutuhkan hanya kecil untuk proses enkripsi dan dekripsi lebih cepat. Oleh karena itu, penulis menggunakan SPECK 128/128 yang berarti ukuran blok dan ukuran kunci sebesar 128 bit. Sehingga masukan test vector yang akan digunakan dapat dilihat pada Gambar 4.2.

```
Input Plaintext 1 :6c61766975716520
Input Plaintext 2 :7469206564616d20
Input k1 :0f0e0d0c0b0a0908
Input k2 :0706050403020100
```

Gambar 4.2 Input *Test Vector* Algoritme SPECK 128/128

Proses yang dilakukan pertama kali dalam SPECK adalah *Key Schedule*. Proses ini akan diulang sebanyak 32 kali. Manualisasi proses *Key Schedule* pada ronde 1-2 dapat dilihat pada Gambar 4.3.

```
---------------------------
Key Scheduling
---------------------------
#####-------
Proses Pencarian Key ke-1
#####-------
Key 1 = 1110000011000000101000001000000001100000010000000100000000


#####-------
Proses Pencarian Key ke-2
#####-------
> Melakukan Fungsi Round x=1084818905618843912 ,y=506097522914230528 ,k=0
 - Menghitung x :
  # Konversi nilai = 1084818905618843912 ke biner
    Hasil = 0000111100001110000011010000110000001011000010100000100100001000
  # Rotasi bit ke kanan sebanyak 8 kali
    Hasil = 0000100000001111000011100000110100001100000010110000101000001001
  .. Nilai X = 1000000011110000111000001101000011000000101100001010000001001
  .. Nilai Y = 1110000011000000101000001000000001100000010000000100000000
  # X OR Y
    Hasil = 1111000101010001001100010001000011110000110100001011000010001
  # Nilai X tidak boleh diatas 18446744073709551615
    Hasil = 1111000101010001001100010001000011110000110100001011000010001
  # X XOR 0
    Hasil = 1111000101010001001100010001000011110000110100001011000010001

 - Menghitung y :
  # Konversi nilai = 506097522914230528 ke biner
    Hasil = 0000011100000110000001010000010000000011000000100000000100000000
  # Rotasi bit ke kiri sebanyak 3 kali
    Hasil = 0011100000110000001010000010000000011000000100000000100000000000
  .. Nilai X = 1111000101010001001100010001000011110000110100001011000010001
  .. Nilai Y = 1110000011000000101000001000000001100000010000000100000000000
  # Y XOR X
    Hasil = 1101110010010100111011001100010001011100011101000000110000010001

Key 2 = 1101110010010100111011001100010001011100011101000000110000010001
```

Gambar 4.3 Manualisasi *Key Schedule* Ronde 1-2

Pada Gambar 4.3, proses *key schedule* pada ronde pertama akan sama dengan input *key* kedua (k2). Sedangkan pada ronde kedua sampai terakhir akan membentuk pola yang sama. Dalam menghitung nilai X, nilai X harus diubah dalam bentuk basis 2 yang kemudian akan dilakukan rotasi bit ke kanan sebanyak 8 kali. Kemudian nilai X akan dilakukan operasi OR dengan nilai Y. Bila nilai X diatas $2^{128} - 1$, maka nilai X perlu dikurangi dengan 2^{128}. Selanjutnya hasil nilai X tersebut akan dilakukan operasi XOR dengan 0 (ronde sekarang - 2). Dalam menghitung nilai Y, nilai Y perlu diubah kedalam basis 2. Kemudian nilai Y akan dirotasi bit ke kiri sebanyak 3 kali. Hasil dari nilai Y tersebut akan dilakukan operasi XOR dengan nilai X. Hasil *key* pada ronde tersebut akan sesuai dengan nilai Y.

Hasil proses *key schedule* dapat dilihat pada Tabel 4.2.

Tabel 4.2 Hasil *Key Schedule*

Key ke-	Hasil
1	11100000110000001010000010000000011000000100000000100000000
2	11011100100101001110110011000100010111000111010000001100001001
3	11111001000111011000100111001100100100000110001000000100001011100
4	11000110101100011111000001111000010100101100110001110110100010001
5	10100111111001101111101001111100111000010110101101111000
6	10110101111110101110000111100100111111100010010011001111111010110
7	10100011011011010110100101010100101100000111001101111100111111110
8	11110101000100010110100100011110101000000010111100110101111100111
9	10100110111010010101011101101110101101000101011010000101010111101
10	10001101110101011111011000100000010011011101110010110010101001010
11	10110010010000111101011111001001100001101001110010101100000011000
12	11101010011111001111010011111000110011001100000010001011001111
13	11110001101011001001000101000111010010110110000111001100011101
14	10000111000101010010101100100011110010111100000010101000110100
15	10101000111111111000101110001100010101001010001110110111110010
16	10010000111001110111110001111000100001110110011111010100111001
17	11101110001111010111111111111001011111100000101110010110001001
18	11101000101001101011100101011110010010110000110001111010010000
19	11100110110000010111010101000101101011100010100100000110010010011
20	10011010111000110110101110000011010110001010010000101001111010
21	11011100011000001011001010101110001001010011000001110000110111
22	10110000000111010000101010111011111000011111101110010111010000
23	11010111100110000111011010000100101000110001100010110101010010
24	10100010001011000101001010000010111001100000000011010011000110
25	11100000001010011101011001111110101111011111100100000000001001000
26	11001110101010110010010001101001100100001001110111111011011111
27	11001010001011100111001111000011001011000000010110100101011100
28	10010011001111000111110001100001111001100000011001010100011001
29	11100010100011011011101101001011011011101011111011011100101010
30	11011110110110010011110011111110001100011011101011100011000000100
31	11000101001111010001100010111001000101110111000010110010011001010
32	10000110011001110010000111000011011011100011101100100100111111

Pada algoritme SPECK 128/128, enkripsi dilakukan perulangan selama 32 kali. Manualisasi proses enkripsi pada ronde 1-2 dapat dilihat pada Gambar 4.4.

```
---------------------------
Enkripsi
---------------------------
Enkripsi dilakukan 32 Round

#####-------
ROUND 1
#####-------
> Melakukan Fungsi Round x=7809653424151160096 ,y=8388271400802151712 ,k=506097522914230528
 - Menghitung x :
  # Konversi nilai = 7809653424151160096 ke biner
    Hasil = 0110110001100001011101100110100101110101011100010110010100100000
  # Rotasi bit ke kanan sebanyak 8 kali
    Hasil = 0010000001101100011000010111011001101001011101010111000101100101
  .. Nilai X = 1000000110110001100001011101100110100101110101011100010110010 1
  .. Nilai Y = 1110100011010010010000001100101011001000110000101101101001000 00
  # X OR Y
    Hasil = 1001010011010101100000011101101111001101110101101101111010000101
  # Nilai X tidak boleh diatas 18446744073709551615
    Hasil = 1001010011010101100000011101101111001101110101101101111010000101
  # X XOR 506097522914230528
    Hasil = 1001001111010011100001001101111111001110110101001101111110000101
 - Menghitung y :
  # Konversi nilai = 8388271400802151712 ke biner
    Hasil = 0111010001101001001000000110010101100100011000010110110100100000
  # Rotasi bit ke kiri sebanyak 3 kali
    Hasil = 1010001101001001000000110010101100100011000010110110100100000011
  .. Nilai X = 1001001111010011100001001101111111001110110101001101111110000101
  .. Nilai Y = 1010001101001001000000110010101100100011000010110110100100000011
  # Y XOR X
    Hasil = 1100001001101010000111111101001110110111011111110110110000110

Hasil X = 93d384dfced4df85
Hasil Y = 309a87f4eddfb686

#####-------
ROUND 2
#####-------
> Melakukan Fungsi Round x=10652003640443985797 ,y=3502261146266613382 ,k=3973647328251544329
 - Menghitung x :
  # Konversi nilai = 10652003640443985797 ke biner
    Hasil = 1001001111010011100001001101111111001110110101001101111110000101
  # Rotasi bit ke kanan sebanyak 8 kali
    Hasil = 1000010110010011110100111000010011011111110011101101010011011111
  .. Nilai X = 1000010110010011110100111000010011011111110011101101010011011111
  .. Nilai Y = 1100001001101010000111111101001110110111011111110110110000110
  # X OR Y
    Hasil = 1011011000101110010110110111100111001101101011101000101101100101
  # Nilai X tidak boleh diatas 18446744073709551615
    Hasil = 1011011000101110010110110111100111001101101011101000101101100101
 - Menghitung y :
  # Konversi nilai = 3502261146266613382 ke biner
    Hasil = 0011000010011010100001111111010011101101110111111101101101000110
  # Rotasi bit ke kiri sebanyak 3 kali
    Hasil = 1000010011010100001111111010011101101110111111101101101000110001
  .. Nilai X = 1000000100001011011000000010010001101101010110011100010000110 1100
  .. Nilai Y = 1000010011010100001111111010011101101110111111101101101000110001
  # Y XOR X
    Hasil = 1011101111101011111111011111011010001001110001111000101110 1

Hasil X = 810b6048dab3886c
Hasil Y = 5df5fefb44e3c5d
```

Gambar 4.4 Manualisasi Enkripsi Ronde 1-2

Pada Gambar 4.4, proses enkripsi pada ronde pertama sampai terakhir akan membentuk pola yang sama. Dalam menghitung nilai X, nilai X harus diubah dalam bentuk basis 2 yang kemudian akan dilakukan rotasi bit ke kanan sebanyak 8 kali. Kemudian nilai X akan dilakukan operasi OR dengan nilai Y. Bila nilai X diatas $2^{128} - 1$, maka nilai X perlu dikurangi dengan 2^{128}. Selanjutnya hasil nilai X tersebut akan dilakukan operasi XOR dengan kunci pada ronde tersebut. Dalam menghitung nilai Y, nilai Y perlu diubah kedalam basis 2. Kemudian nilai Y akan dirotasi bit ke kiri sebanyak 3 kali. Hasil dari nilai Y tersebut akan dilakukan operasi XOR dengan nilai X. Nilai X akan menjadi nilai ciphertext 1 sedangkan nilai Y akan menjadi nilai ciphertext 2. Hasil proses enkripsi dapat dilihat pada Tabel 4.3.

Tabel 4.3 Hasil Enkripsi

Ronde Ke-	Hasil Ciphertext 1 (X)	Hasil Ciphertext 2 (Y)
1	93d384dfced4df85	309a87f4eddfb686
2	810b6048dab3886c	5df5fefb44e3c5d
3	8b7de2836dece7b9	a5871dfecf9d0551
4	99a36b9901c684b1	b59b846f7d2eae3c
5	667aea2feff2a230	caa6c9540687d3d5
6	4ef7a5dac85309a1	1bc1ef7afc6d970f
7	1e7d8e74674696e6	c072f5a3842a2e9e
8	53801a2f58be40c7	5017b73379ef3431
9	441f9cfaf36cb72c	c4a225613c1516a6
10	7d33b2de7ad431f8	582299d79a7c84ce
11	e2dc1a43fe6bf4e7	23c8d4ff2d8fd295
12	7e95cb6517ee7b17	60d36c9c7b90efbe
13	844ac445183802	61f2e27999f45f1
14	8f0a99519624f6fb	bff3e86d5aded973
15	137d788af8d7489b	ece23be02e218306
16	cf860764faa9b037	a897d8658ba5a800
17	9779e1904fa59aa3	d3c722bc1288daa6
18	9ff82032875ebd60	1c135d213186856
19	84a3c77919cdcb80	8aaa69e9810e8930
20	465eb871567a420e	130df73d5e0e0b8a
21	fd34e75bea54f510	655b5eb11a24a940
22	c645992397f56974	ec9f6cab46d02377
23	b6fdc4c0c970adaa	d206a19afff1b615
24	de91cddd26bbf5db	4ea4c10ad9364575
25	caaa84a60ba40122	bf8c8cf0c2162a88
26	8502a541a06f3336	7966c2c7b0de6773
27	cafcf99c397fbffa	1caefa1bf8c8461
28	d85af38322479139	d60d8e8ede23b231
29	37468750baea4ee8	872af3264bf7df66
30	b1bb0553ad082ab0	88ec9c61f2b6d184
31	fca34fde51136bcb	bbc7acd1c4a5e7ef
32	a65d985179783265	7860fedf5c570d18

Pada algoritme SPECK 128/128, dekripsi dilakukan secara kebalik, yaitu dari ronde 32 ke ronde 1. Manualisasi proses dekripsi 2 ronde pertama dapat dilihat pada Gambar 4.5.

```
---------------------------
Dekripsi
---------------------------
Dekripsi dilakukan 32 Round

#####-------
ROUND 32
#####-------
> Melakukan Fungsi Round x=11987905258827821669 ,y=8674213117595946264 ,k=2421186661733812543
 - Menghitung y :
  .. Nilai X = 1010011001011101100110000101000101111001011110000011001001100101
  .. Nilai Y = 1111000011000001111111101101111101011100010101110000110100011000
  # Y XOR X
    Hasil = 1101111000111101011001101000111000100101001011110011111101111101
  # Konversi nilai = 16014068610694594429 ke biner
    Hasil = 1101111000111101011001101000111000100101001011110011111101111101
  # Rotasi bit ke kanan sebanyak 3 kali
    Hasil = 1011101111000111101011001101001111000100101001011110011111101111
 - Menghitung x :
  .. Nilai X = 1010011001011101100110000101000101111001011110000011001001100101
  .. Nilai Y = 1011101111000111101011001101000111000100101001011110011111101111
  # X XOR 2421186661733812543
    Hasil = 1000011111000100010100000010000110100010111101101111101011011010
  # X - Y
    Hasil = -1101000000001101011100101100000010000110101110111011010010101
  # Nilai X tidak boleh dibawah 0
    Hasil = 1100101111111100101000110100111111011111001010001000100110110101011
  # Konversi nilai = 14698802847258055531 ke biner
    Hasil = 1100101111111100101000110100111111011111001010001000100110110101011
  # Rotasi bit ke kiri sebanyak 8 kali
    Hasil = 1111110010100011010011111101111001010001000100110110101111001011
Hasil X = fca34fde51136bcb
Hasil Y = bbc7acd1c4a5e7ef

#####-------
ROUND 31
#####-------
> Melakukan Fungsi Round x=18204481935023238091 ,y=13530973622405294063 ,k=1421254318229243146
 - Menghitung y :
  .. Nilai X = 1111110010100011010011111101111001010001000100110110101111001011
  .. Nilai Y = 1011101111000111101011001101000111000100101001011110011111101111
  # Y XOR X
    Hasil = 1000111011001001110001100001111100101011011011010001100001000100
  # Konversi nilai = 5144486330439732260 ke biner
    Hasil = 0100011101100100111000110000111110010101101101101000110000100100
  # Rotasi bit ke kanan sebanyak 3 kali
    Hasil = 1000100011101100100111000110000111110010101101101101000110000100

 - Menghitung x :
  .. Nilai X = 1111110010100011010011111101111001010001000100110110101111001011
  .. Nilai Y = 1000100011101100100111000110000111110010101101101101000110000100
  # X XOR 14212543182292431461
    Hasil = 1110011001111001010111011001110100011001100011110110011010111
  # X - Y
    Hasil = -1001111010011100100010011111101010101100010100101111011111010110
  # Nilai X tidak boleh dibawah 0
    Hasil = 1011000010110001101110110000010101010011101011010000100000101010
  # Konversi nilai = 12732163253106182186 ke biner
    Hasil = 1011000010110001101110110000010101010011101011010000100000101010
  # Rotasi bit ke kiri sebanyak 8 kali
    Hasil = 1011000110111011000001010101001110101101000010000010101010110000
Hasil X = b1bb0553ad082ab0
Hasil Y = 88ec9c61f2b6d184
```

Gambar 4.5 Manualisasi Dekripsi dengan 2 Ronde Pertama

Pada Gambar 4.5, proses dekripsi pada ronde pertama sampai terakhir akan membentuk pola yang sama. Dalam menghitung nilai Y, nilai Y akan dilakukan operasi XOR dengan nilai X. Kemudian hasilnya akan dirotasi bit ke kanan sebanyak 3 kali. Dalam menghitung nilai X, nilai X akan dilakukan operasi XOR dengan ronde tersebut. Kemudian nilai X akan dilakukan operasi SUBSTRACT dengan nilai Y. Bila nilai X dibawa 0, maka nilai X perlu ditambah dengan 2^{128}. Selanjutnya hasil nilai X tersebut akan dilakukan rotasi bit ke kiri sebanyak 8 kali. Nilai X akan menjadi nilai decrypted 1 sedangkan nilai Y akan menjadi nilai decrypted 2. Hasil proses enkripsi dapat dilihat pada Tabel 4.4.

Tabel 4.4 Hasil Dekripsi

Ronde Ke-	Hasil Dekripsi 1 (X)	Hasil Dekripsi 2 (Y)
32	fca34fde51136bcb	bbc7acd1c4a5e7ef
31	b1bb0553ad082ab0	88ec9c61f2b6d184
30	37468750baea4ee8	872af3264bf7df66
29	d85af38322479139	d60d8e8ede23b231
28	cafcf99c397fbffa	1caefa1bf8c8461
27	8502a541a06f3336	7966c2c7b0de6773
26	caaa84a60ba40122	bf8c8cf0c2162a88
25	de91cddd26bbf5db	4ea4c10ad9364575
24	b6fdc4c0c970adaa	d206a19afff1b615
23	c645992397f56974	ec9f6cab46d02377
22	fd34e75bea54f510	655b5eb11a24a940
21	465eb871567a420e	130df73d5e0e0b8a
20	84a3c77919cdcb80	8aaa69e9810e8930
19	9ff82032875ebd60	1c135d213186856
18	9779e1904fa59aa3	d3c722bc1288daa6
17	cf860764faa9b037	a897d8658ba5a800
16	137d788af8d7489b	ece23be02e218306
15	8f0a99519624f6fb	bff3e86d5aded973
14	844ac445183802	61f2e27999f45f1
13	7e95cb6517ee7b17	60d36c9c7b90efbe
12	e2dc1a43fe6bf4e7	23c8d4ff2d8fd295
11	7d33b2de7ad431f8	582299d79a7c84ce
10	441f9cfaf36cb72c	c4a225613c1516a6
9	53801a2f58be40c7	5017b73379ef3431
8	1e7d8e74674696e6	c072f5a3842a2e9e
7	4ef7a5dac85309a1	1bc1ef7afc6d970f
6	667aea2feff2a230	caa6c9540687d3d5
5	99a36b9901c684b1	b59b846f7d2eae3c
4	8b7de2836dece7b9	a5871dfecf9d0551
3	810b6048dab3886c	5df5fefb44e3c5d
2	93d384dfced4df85	309a87f4eddfb686
1	6c61766975716520	7469206564616d20

Hasil dari perancangan algoritme SPECK dapat dilihat pada Gambar 4.6.

Plaintext 1 = 6c61766975716520
Plaintext 2 = 7469206564616d20

Kunci 1 = f0e0d0c0b0a0908
Kunci 2 = 706050403020100

Ciphertext 1 = a65d985179783265
Ciphertext 2 = 7860fedf5c570d18

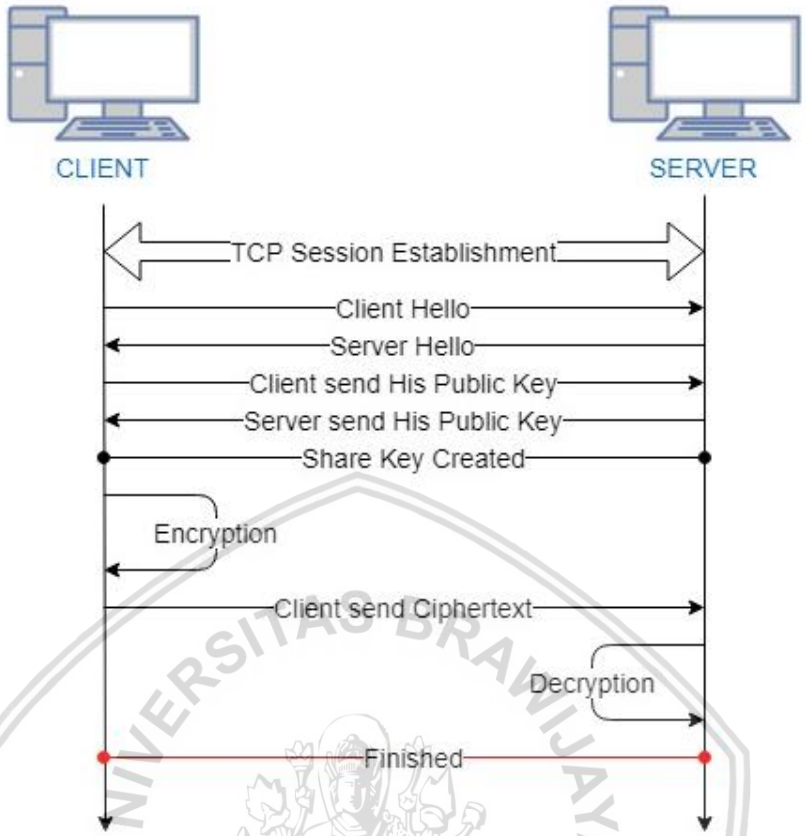
Hasil Dekripsi 1 = 6c61766975716520
Hasil Dekripsi 2 = 7469206564616d20

Waktu eksekusi *Key Scheduling* : 291.0 ms
Waktu eksekusi Enkripsi : 415.0 ms
Waktu eksekusi Dekripsi : 485.0 ms

Gambar 4.6 Hasil *Test Vector*

4.2.2 Perancangan Sistem

Penulis membuat sistem pengiriman rekaman audio melalui jaringan internet menggunakan algoritme *block cipher*, yaitu SPECK. Versi SPECK yang digunakan adalah ukuran blok 128 bit dengan besar ukuran kunci 128 bit. Hal ini dikarenakan rekaman audio memiliki ukuran bit yang besar. Selain itu, keamanan dalam melakukan enkripsi secara otomatis meningkat bila menggunakan ukuran kunci yang semakin besar. Dalam pertukaran kunci antara klien dengan server, algoritme yang digunakan adalah algoritme *Diffie-Hellman key exchange* (DHKE). Algoritme DHKE berfungsi untuk menjaga keamanan proses pertukaran kunci di suatu jaringan. Pengiriman data antar server dan klien menggunakan pemrograman *socket*.



Gambar 4.7 Gambaran Umum Sistem

Pada Gambar 4.7, dapat dilihat gambaran umum sistem penelitian ini. Pertama-tama, server dijalankan terlebih dahulu, sehingga menandakan server siap menerima data dari klien. Kemudian klien berusaha untuk mengkoneksi server dan server akan menyetujui koneksi dari klien. Kedua, proses pertukaran kunci menggunakan algoritme DHKE. Ketiga, klien akan melakukan enkripsi SPECK dengan menggunakan hasil kunci dari proses DHKE. Keempat, klien mengirimkan *ciphertext*, hasil enkripsi rekaman audio. Kelima, server melakukan dekripsi *ciphertext* menjadi *plaintext*. Hasil plaintext akan menjadi rekaman audio yang sesuai dengan klien miliki.

4.2.3 Perancangan Pengujian

Pada tahap ini, penulis akan melakukan beberapa pengujian kepada system yang telah dibuat. Pengujian yang dilakukan oleh penulis ada 6, yaitu pengujian test vector, pengujian fungsional, pengujian validitas enkripsi dan dekripsi, pengujian waktu enkripsi dan dekripsi dengan variasi ekstensi rekaman, pengujian waktu sistem, dan pengujian keamanan dengan wireshark.

1. Pengujian Test Vector

   Pengujian ini berfungsi untuk memastikan algoritme SPECK penulis sesuai dengan algoritme SPECK di jurnal "*The Simon and SPECK Families of Lightweight Block Clphers*"

2. Pengujian Fungsional

   Pengujian ini bertujuan untuk memastikan semua fitur dalam sistem dapat berjalan sesuai fungsinya.

3. Pengujian Validitas Enkripsi dan Dekripsi

   Pengujian ini bertujuan untuk memastikan plainteks sama dengan hasil dekripsinya. Pengujian ini akan menggunakan *input* rekaman audio dengan variasi ekstensi *file* dan durasi.

4. Pengujian Waktu Enkripsi dan Dekripsi

   Pengujian ini berfungsi untuk mengetahui waktu enkripsi dan dekripsi dari algoritme SPECK. Pengujian ini juga menggunakan *input* rekaman audio dengan variasi ekstensi *file* dan durasi.

5. Pengujian Waktu Sistem

   Pengujian ini bertujuan untuk membandingkan waktu eksekusi algoritme SPECK dengan variasi ekstensi *file* pada rekaman audio.

6. Pengujian Keamanan

   Pengujian ini berfungsi untuk mengetahui tingkat keamanan bila diserang *man in the middle* pasif.

BAB 5 IMPLEMENTASI

Bab ini berisi kebutuhan-kebutuhan apa saja yang akan digunakan dalam penelitian beserta rancangan penelitian ini baik rancangan algoritme, sistem dan pengujian yang akan dilakukan.

5.1 Algoritme SPECK

Dalam penelitian ini, algoritme SPECK yang digunakan adalah SPECK dengan ukuran blok dan ukuran kunci sebesar 128 bit. *Source code* inisialisasi awal algoritme SPECK dapat dilihat pada Algoritme 1.

Algoritme 1 : Inisialisasi Awal Algoritme SPECK

```
1  static int m = 2; //key words
2  static int T = 32; //rounds
3  static int alpha = 8; //alpha
4  static int beta = 3; //beta
5  static String limitString = "18446744073709551615";
6  static String limitPlus1String = "18446744073709551616";
```

5.1.1 Fungsi Rotasi Kiri

Fungsi rotasi kiri akan digunakan dalam proses *key schedule*, enkripsi, maupun dekripsi. *Source code* rotasi kiri dapat dilihat pada Algoritme 2.

Algoritme 2 : Fungsi Rotasi Kiri

```
1  public static BigInteger rotateLeft(BigInteger number, int amount)
   {
2     String str = number.toString(2);
3     char[] binaryFromNumber = str.toCharArray();
4     char[] binary = new char[64];
5     int i,j;
6     //Menambah bit 0 di depan
7     for(j=binary.length-1; j>=0; j--){
8        if(binary.length-binaryFromNumber.length <= j)
9           binary[j] = binaryFromNumber[binaryFromNumber.
   length - (binary.length-j)];
10       else
11          binary[j] = '0';
12    }
13    //rotate left
14    char temp;
15    for(i=1; i<=amount; i++){
16       temp = binary[0];
17       for(j=0; j<=binary.length-2; j++){
18          binary[j] = binary[j+1];
19       }
20       binary[binary.length-1] = temp;
21    }
22    str = String.valueOf(binary);
23    BigInteger result = new BigInteger(str, 2);
24    return result;
25 }
```

5.1.2 Fungsi Rotasi Kanan

Fungsi rotasi kanan akan digunakan dalam proses *key schedule*, enkripsi, maupun dekripsi. *Source code* rotasi kanan dapat dilihat pada Algoritme 3.

Algoritme 3 : Fungsi Rotasi Kanan

```
1   public static BigInteger rotateRight(BigInteger number, int
    amount) {
2      String str = number.toString(2);
3      char[] binaryFromNumber = str.toCharArray();
4      char[] binary = new char[64];
5      int i,j;
6      //Menambah bit 0 di depan
7      for(j=binary.length-1; j>=0; j--){
8         if(binary.length-binaryFromNumber.length <= j)
9            binary[j] = binaryFromNumber[binaryFromNumber.
    length - (binary.length-j)];
10        else
11           binary[j] = '0';
12     }
13     //rotate Right
14     char temp;
15     for(i=1; i<=amount; i++){
16        temp = binary[binary.length-1];
17        for(j=binary.length-1; j>=1; j--){
18           binary[j] = binary[j-1];
19        }
20        binary[0] = temp;
21     }
22     str = String.valueOf(binary);
23     BigInteger result = new BigInteger(str, 2);
24     return result;
25  }
```

5.1.3 Fungsi SPECK *Round*

Fungsi SPECK *Round* digunakan dalam proses *key schedule* dan enkripsi. *Source code* SPECK *Round* dapat dilihat pada Algoritme 4.

Algoritme 4 : Fungsi Round SPECK

```
1   public static BigInteger[] speckRound(BigInteger x, BigInteger y,
    BigInteger k){
2      x = rotateRight(x,alpha);
3      x = x.add(y);
4      BigInteger limit = new BigInteger(limitString);
5      BigInteger limitPlus1 = new BigInteger(limitPlus1String);
6      while (x.compareTo(limit) > 0){
7         x = x.subtract(limitPlus1);
8      }
9      x = x.xor(k);
10
11     y = rotateLeft(y, beta);
12     y = y.xor(x);
13
14     BigInteger[] result = new BigInteger[2];
15     result[0] = x;
16     result[1] = y;
17     return result;
18  }
```

5.1.4 Fungsi SPECK Reverse *Round*

Fungsi SPECK *Reverse Round* hanya digunakan dalam proses dekripsi. *Source code* SPECK *Reverse Round* dapat dilihat pada Algoritme 5.

Algoritme 5 : Fungsi Reverse Round SPECK

```
1   public static BigInteger[] speckReverseRound(BigInteger x,
    BigInteger y, BigInteger k){
2      y = x.xor(y);
3      y = rotateRight(y, beta);
4
5      x = x.xor(k);
6      x = x.subtract(y);
7      BigInteger zero = new BigInteger("0");
8      BigInteger limitPlus1 = new BigInteger(limitPlus1String);
9      while (x.compareTo(zero) < 0){
10        x = x.add(limitPlus1);
11     }
12     x = rotateLeft(x, alpha);
13
14     BigInteger[] result = new BigInteger[2];
15     result[0] = x;
16     result[1] = y;
17     return result;
18  }
```

5.1.5 Fungsi Key Scheduling

Fungsi *Key Scheduling* berfungsi untuk membuat kunci sebanyak ronde sesuai versi SPECK yang digunakan. Pada SPECK 128/128, rondenya sebanyak 32 kali. Kunci-kunci ini akan digunakan untuk melakukan enkripsi dan dekripsi. *Source code key scheduling* dapat dilihat pada Algoritme 6.

Algoritme 6 : Fungsi Key Scheduling

```
1   public static BigInteger[] keyScheduling(BigInteger[] key){
2      BigInteger[] result = new BigInteger[T];
3      BigInteger[] temp = new BigInteger[2];
4      BigInteger k1 = key[0];
5      BigInteger k2 = key[1];
6
7      result[0] = k2;
8      for (int i = 0; i < T-1; i++) {
9         String str = String.valueOf(i);
10        BigInteger j = new BigInteger(str);
11        temp = speckRound(k1, k2, j);
12        k1 = temp[0];
13        k2 = temp[1];
14        result[i+1] = k2;
15     }
16     return result;
17  }
```

5.1.6 Fungsi Enkripsi

Fungsi enkripsi akan dilakukan sebanyak 32 kali, karena ronde pada SPECK 128/128 sebanyak 32 kali. *Source code* enkripsi dapat dilihat pada Algoritme 7.

Algoritme 7 : Fungsi Enkripsi

```
1   public static BigInteger[] encryption(BigInteger[] plaintext,
    BigInteger[] keySchedule){
2      BigInteger[] ciphertext = new BigInteger[2];
3      ciphertext[0] = plaintext[0];
```

```
4       ciphertext[1] = plaintext[1];
5       for(int i=0; i<= T-1; i++){
6          ciphertext = speckRound(ciphertext[0], ciphertext[1],
        keySchedule[i]);
7       }
8       return ciphertext;
9    }
```

5.1.7 Fungsi Dekripsi

Fungsi dekripsi akan dilakukan sebanyak 32 kali, karena ronde pada SPECK 128/128 sebanyak 32 kali. *Source code* dekripsi dapat dilihat pada Algoritme 8.

```
Algoritme 8 : Fungsi Dekripsi
1    public static BigInteger[] decryption(BigInteger[] ciphertext,
     BigInteger[] keySchedule){
2       BigInteger[] plaintext = new BigInteger[2];
3       plaintext[0] = ciphertext[0];
4       plaintext[1] = ciphertext[1];
5       for(int i=T-1; i>=0; i--){
6          plaintext = speckReverseRound(plaintext[0], plaintext[1],
        keySchedule[i]);
7       }
8       return plaintext;
9    }
```

5.2 Sistem Klien

5.2.1 *Socket* Programming

Penulis menggunakan pemrograman *socket* agar server dan klien dapat saling bertukar data. Klien menghubungi server dengan *IP address* 192.168.56.101 dan port 1234. *Source code* pemrograman *socket* pada klien dapat dilihat pada Algoritme 9.

```
Algoritme 9 : Pemrograman Socket pada Klien
1    Scanner inSc = new Scanner(System.in);
2    Scanner inScLine = new Scanner(System.in);
3    System.out.println("--Menghubungi Server.......");
4    Socket socket=new Socket("192.168.56.101",1234);
5    System.out.println("--Server telah terkoneksi.......");
6
7    DataInputStream din=new
        DataInputStream(socket.getInputStream());
8    DataOutputStream dout=new
        DataOutputStream(socket.getOutputStream());
9    BufferedReader br=new BufferedReader(new
        InputStreamReader(System.in));
```

5.2.2 Algoritme DHKE

Penulis menggunakan algoritme DHKE agar klien dan server memiliki kunci rahasia yang sama untuk melakukan enkripsi dan dekripsi. Hal ini sangat diperlukan agar orang ketiga tidak mengetahui kunci untuk melakukan dekripsi. Prosesnya adalah pertama, klien menginputkan *private key* yang akan diproses dengan rumus DKHE menjadi *public key* milik klien. Kemudian, klien akan mengirimkan *public key* tersebut ke server. Proses yang sama akan dilakukan oleh server, sampai pada akhirnya server mengirimkan *public key* miliknya ke klien. Terakhir, klien dan server akan melakukan rumus DHKE agar mendapatkan

shared key. *Shared key* pada server dan klien akan pasti sama. *Shared key* yang didapat akan sebesar 128 bit. *Source code* DHKE pada klien dapat dilihat pada Algoritme 10.

Algoritme 10 : DHKE pada Klien

```
1   BigInteger secretA;
2   BigInteger generatorValue, primeValue, publicA, publicB,
       shared;
3   String str;
4   primeValue = new
       BigInteger("330273671788511023796232738497605972489");
5   generatorValue = new BigInteger("2001");
6   System.out.print("Input Secret Key Client (max. 16
       karakter) = ");
7   str = inScLine.nextLine();
8
9   //kata2 to binary
10  String resultBinary = "";
11  char[] strChar = str.toCharArray();
12  int lengthBinary;
13  for (int i = 0; i < strChar.length; i++) {
14     str = Integer.toBinaryString(strChar[i]);
15     lengthBinary = str.length();
16     for (int j = 1; j <= 8 - lengthBinary; j++){
17        str = "0" + str;
18     }
19     resultBinary = resultBinary + str;
20  }
21  secretA = new BigInteger(resultBinary,2);
22  System.out.println("Secret Key Client = "+secretA);
23
24  publicA = generatorValue.modPow(secretA, primeValue);
25  System.out.println("Public key Client = "+publicA);
26  dout.writeUTF(publicA.toString());
27  dout.flush();
28
29  System.out.println("Menunggu Server mengirim Public Key
       Server..");
30  str = din.readUTF();
31
32  publicB = new BigInteger(str);
33  System.out.println("Public key Server = "+publicB);
34
35  shared = publicB.modPow(secretA,primeValue);
36  System.out.println("Shared key = "+shared);
```

5.2.3 Split Shared Key

Kemudian *shared key* tersebut akan dibagi menjadi 2, masing-masing 64 bit. Hal ini dikarenakan algoritme SPECK membutuhkan 2 kunci, masing-masing 64 bit. *Shared key* tersebut akan diubah kedalam bentuk basis 2. Kemudian hasil biner *shared key* dilakukan penambahan "0" di paling kiri agar menjadi 128 bit. Terakhir, dibagi 2 bagian menjadi masing-masing 64 bit. *Source code* memecah *shared key* dapat dilihat pada Algoritme 11.

Algoritme 11 : Split Shared Key pada Klien dan Server

```
1   public static BigInteger[] splitSharedKey(BigInteger
       number){
2      BigInteger result[] = new BigInteger[2];
3      String str = number.toString(2);
4      int i,lengthStr;
5      lengthStr = str.length();
```

```
6       for(i=0; i<128-lengthStr; i++){
7          str="0"+str;
8       }
9
10      result[0] = new BigInteger(str.substring(0, 64),2);
11      result[1] = new BigInteger(str.substring(64),2);
12
13      return result;
14   }
```

5.2.4 Membaca Rekaman Audio

Klien perlu membaca rekaman audio yang nantinya akan dienkripsi dan dikirimkan ke server. Rekaman audio ini akan diubah ke dalam *array byte*. *Source code* membaca rekaman audio dapat dilihat pada Algoritme 12.

Algoritme 12 : Membaca Rekaman Audio

```
1   File file = new File("C:\\Documents and
       Settings\\SkripsiDavid-Client\\My
       Documents\\NetBeansProjects\\SkripsiDavid
       Client\\src\\Client\\tes.wav");
2   int lengthAudio = (int)file.length();
3   byte[] data = new byte[lengthAudio];
4
5   FileInputStream in = new FileInputStream(file);
6   in.read(data);
7   in.close();
```

5.2.5 Mengubah *Array Byte* ke *Array Word*

Di dalam algoritme SPECK ada istilah *word*. Dalam penelitian ini, penulis menggunakan SPECK 128/128. Ukuran *word* pada SPECK 128/128 adalah 64 bit. Karena *byte* berukuran 8 bit, maka dari itu 8 bit ini perlu diubah ke 64 bit. *Source code* mengubah *array byte* ke *array word* dapat dilihat pada Algoritme 13.

Algoritme 13 : Mengubah Array Byte ke Array Word

```
1   public static BigInteger[] byteToWord(byte[] data){
2      int i,j,k;
3      int[] dataInt = new int[data.length];
4      String[] binary = new String[data.length];
5      String str;
6      BigInteger temp;
7      for(i=0; i<data.length; i++){
8         dataInt[i] = data[i] + 128;
9         str = Integer.toString(dataInt[i]);
10        temp = new BigInteger(str);
11        binary[i] = temp.toString(2);
12        k = 8 - binary[i].length();
13        for(j=1; j<=k; j++){
14           binary[i]="0"+binary[i];
15        }
16     }
17
18     //1 word 64 bit
19     int combine= 64/8;
20     int resultLength = (int)Math.floor((double)data.length /
       combine);
21     if(data.length % combine != 0){
22        resultLength = resultLength + 1;
23     }
24     int remain = data.length;
25     boolean cek = false;
26     //supaya selalu genap
```

```
27      if(resultLength%2 != 0){
28         resultLength = resultLength+1;
29         cek = true;
30      }
31      BigInteger[] result = new BigInteger[resultLength];
32
33      for(i=0; i<resultLength; i++){
34         if((cek) && (remain <= 0)){
35            result[i]=new BigInteger("0");
36         } else{
37            if(remain >= 8){
38               combine = 8;
39               remain = remain - combine;
40            }else{
41               combine = remain;
42               remain = remain - combine;
43            }
44            str = "";
45            for(j=i*8; j<i*8+combine; j++){
46               str = str + binary[j];
47            }
48            result[i] = new BigInteger(str,2);
49         }
50      }
51      return result;
52   }
```

5.2.6 Proses *Key Schedule*

Key yang telah dibagi menjadi 2 akan menjadi variabel fungsi *key scheduling*. Key Schedule digunakan untuk membuat kunci pada setiap ronde dalam melakukan enkripsi dan dekripsi. *Source code key schedule* dapat dilihat pada Algoritme 14.

Algoritme 14 : Key Schedule pada Klien dan Server

```
1   BigInteger[] keySchedule = new BigInteger[32];
2   keySchedule = Speck128.keyScheduling(k);
```

5.2.7 Melakukan Enkripsi dan Mengirimkan ke Server

Klien akan mengirimkan panjang rekaman audio dan panjang *array ciphertext*. Hal ini bertujuan untuk mempermudah Server dalam mengubah *array* tersebut ke rekaman kembali. Kemudian klien akan melakukan enkripsi dan hasil tersebut akan langsung dikirim ke server dalam perulangan. *Source code* enkripsi dan mengirimkan ke server dapat dilihat pada Algoritme 15.

Algoritme 15 : Enkripsi dan Mengirimkan ke Server

```
1   System.out.println("--Send Length and Array Ciphertext");
2   System.out.println("Length Audio = "+lengthAudio);
3   dout.writeInt(lengthAudio);
4   dout.flush();
5
6   int i=0;
7   BigInteger[] plaintext = new BigInteger[2];
8   BigInteger[] ciphertext = new BigInteger[2];
9   System.out.println("Length Array Ciphertext =
       "+word.length);
10  dout.writeInt(word.length);
11  dout.flush();
12
13  System.out.println("--Encript");
14  //encrypt + send cipher
```

```
15  while(i<word.length){
16     plaintext[0] = word[i];
17     plaintext[1] = word[i+1];
18     ciphertext = Speck128.encryption(plaintext, keySchedule);
       str = ciphertext[0].toString();
19     dout.writeUTF(str);
20     dout.flush();
21     str = ciphertext[1].toString();
22     dout.writeUTF(str);
23     dout.flush();
24     i=i+2;
25  }
26  dout.close();
27  socket.close();
28  System.out.println("--Done");
29
```

5.3 Sistem Server

5.3.1 Socket Programming

Penulis menggunakan pemrograman *socket* agar server dan klien dapat saling bertukar data. Server membuka koneksi dengan port telah ditentukan, yaitu 1234. *IP address* server adalah 192.168.56.101. Server menunggu klien untuk melakukan koneksi dengan server. *Source code* pemrograman *socket* pada server dapat dilihat pada Algoritme 16.

Algoritme 16 : Pemrograman Socket pada Server

```
1   Scanner inSc = new Scanner(System.in);
2   Scanner inScLine = new Scanner(System.in);
3   ServerSocket serversocket=new ServerSocket(1234);
4   System.out.println("--Menunggu Koneksi.......");
5   Socket socket=serversocket.accept();
6   System.out.println("--Koneksi Client telah
       disetujui.......");
7
8   DataInputStream din=new
       DataInputStream(socket.getInputStream());
9   DataOutputStream dout=new
       DataOutputStream(socket.getOutputStream());
10  BufferedReader br=new BufferedReader(new
       InputStreamReader(System.in));
```

5.3.2 Algoritme DHKE

Penulis menggunakan algoritme DHKE agar klien dan server memiliki kunci rahasia yang sama untuk melakukan enkripsi dan dekripsi. Hal ini sangat diperlukan agar orang ketiga tidak mengetahui kunci untuk melakukan dekripsi. Prosesnya adalah pertama, klien menginputkan *private key* yang akan diproses dengan rumus DKHE menjadi *public key* milik klien. Kemudian, klienk akan mengirimkan *public key* tersebut ke server. Proses yang sama akan dilakukan oleh server, sampai pada akhirnya server mengirimkan *public key* miliknya ke klien. Terakhir, klien dan server akan melakukan rumus DHKE agar mendapatkan *shared key*. *Shared key* pada server dan klien akan pasti sama. *Shared key* yang didapat akan sebesar 128 bit. *Source code* DHKE pada server dapat dilihat pada Algoritme 17.

Algoritme 17 : DHKE pada Server

```
1   BigInteger secretB;
2   BigInteger generatorValue, primeValue, publicA, publicB,
       shared;
3   String str;
4   primeValue = new
       BigInteger("330273671788511023796232738497605972489");
5   generatorValue = new BigInteger("2001");
6   System.out.println("Menunggu Client mengirim Public Key
       Client..");
7   str = din.readUTF();
8   publicA = new BigInteger(str);
9   System.out.println("Public key Client = "+publicA);
10
11  System.out.print("Input Secret Key Server (max. 16
       karakter) = ");
12  str = inScLine.nextLine();
13
14  //kata2 to binary
15  String resultBinary = "";
16  char[] strChar = str.toCharArray();
17  int lengthBinary;
18  for (int i = 0; i < strChar.length; i++) {
19     str = Integer.toBinaryString(strChar[i]);
20     lengthBinary = str.length();
21     for (int j = 1; j <= 8 - lengthBinary; j++){
22        str = "0" + str;
23     }
24     resultBinary = resultBinary + str;
25  }
26  secretB = new BigInteger(resultBinary,2);
27  System.out.println("Secret Key Server = "+secretB);
28
29  publicB = generatorValue.modPow(secretB, primeValue);
30  System.out.println("Public key Server = "+publicB);
31  dout.writeUTF(publicB.toString());
32  dout.flush();
33
34  shared = publicA.modPow(secretB,primeValue);
35  System.out.println("Shared key = "+shared);
```

5.3.3 Split Shared Key

Pemecahan *shared key* menjadi 2 dengan masing-masing 64 bit pada server sama persis dengan *source code* di klien. Proses pemecahan *shared key* dapat dilihat pada sub bab 5.2.3.

5.3.4 Menerima Data dari Klien

Server akan menerima data panjang rekaman audio dan panjang *ciphertext*. Kemudian server akan menerima *array ciphertext* dari klien. *Source code* menerima data dari klien dapat dilihat pada Algoritme 18.

Algoritme 18 : Menerima Data dari Klien

```
1   System.out.println("--Receive Length Audio and Array
       Ciphertext");
2   int lengthAudio = din.readInt();
3   System.out.println("Length Audio = "+lengthAudio);
4   int lengthCipher = din.readInt();
5   System.out.println("Length Array Ciphertext ="
       +lengthCipher);
6   BigInteger[] resultCipher = new BigInteger[lengthCipher];
7   int i;
```

```
8    for(i=0;i<lengthCipher;i++){
9       str = din.readUTF();
10      resultCipher[i] = new BigInteger(str);
11   }
```

5.3.5 Proses *Key Schedule*

Key yang telah dibagi menjadi 2 akan menjadi variabel fungsi *key scheduling*. *Key schedule* digunakan untuk membuat kunci pada setiap ronde dalam melakukan enkripsi dan dekripsi. Proses *key schedule* pada server sama persis dengan klien. Proses *key schedule* dapat dilihat pada sub bab 5.2.6.

5.3.6 Melakukan Dekripsi

Server melakukan dekripsi dari *ciphertext* yang diterima dari klien. Kunci yang digunakan dari proses hasil *key schedule* sebelumnya. *Source code* memecah *shared key* dapat dilihat pada Algoritme 19.

```
Algoritme 19 : Dekripsi
1    System.out.println("--Decrypt");
2    i=0;
3    BigInteger[] ciphertext = new BigInteger[2];
4    BigInteger[] plaintext = new BigInteger[2];
5    BigInteger[] resultDecrypt = new BigInteger[lengthCipher];
6    while(i<lengthCipher){
7       ciphertext[0] = resultCipher[i];
8       ciphertext[1] = resultCipher[i+1];
9       plaintext = Speck128.decryption(ciphertext,
     keySchedule);
10      resultDecrypt[i]=plaintext[0];
11      resultDecrypt[i+1]=plaintext[1];
12      i=i+2;
13   }
```

5.3.7 Mengubah *Array Word* ke *Array Byte*

Kemudian server akan mengubah *array word* (hasil dari dekripsi) akan diubah ke *array byte*. *Word* dalam algoritme SPECK 128/128 sebesar 64 bit. Sedangkan *byte* memiliki ukuran sebesar 8 bit. *Source code* mengubah *array word* ke *array byte* dapat dilihat pada Algoritme 20.

```
Algoritme 20 : Mengubah Array Word ke Array Byte
1    public static byte[] wordToByte(BigInteger[] data, int
     byteLength){
2       byte[] result = new byte[(int)byteLength];
3       String[] binary = new String[data.length];
4       String str;
5       int i,j,k;
6       for(i=0; i<data.length; i++){
7          binary[i]=data[i].toString(2);
8       }
9       int remain, combine=8;
10      remain=byteLength;
11
12      i=0;
13      while(remain>0){
14         if(remain >= 8){
15            combine = 8;
16            remain = remain - combine;
17         }else{
18            combine = remain;
```

```
19            remain = remain - combine;
20         }
21         k = combine*8 - binary[i].length();
22         for(j=1; j<=k; j++){
23            binary[i]="0"+binary[i];
24         }
25         k=0;
26
27         for(j=i*8; j<i*8+combine; j++){
28            if(j<i*8+combine-1){
29               str = binary[i].substring(k, k+8);
30            }else{
31               str = binary[i].substring(k);
32            }
33
34            k = k + 8;
35            result[j] = (byte)(Integer.valueOf(str, 2) - 128);
36         }
37         i = i + 1;
38      }
39      return result;
40   }
```

5.3.8 Mengubah *Array Byte* ke Rekaman Audio

Tahap terakhir adalah server mengubah *array byte* menjadi rekaman audio kembali. Setelah server melakukan proses pengubahan ke rekaman audio, server akan menutup koneksi *socket*. *Source code* memecah *shared key* dapat dilihat pada Algoritme 21.

Algoritme 21 : Mengubah Array Byte ke Rekaman Audio

```
1   System.out.println("--Make Audio");
2   File file = new File("C:\\Documents and
       Settings\\SkripsiDavid-Server\\My
       Documents\\NetBeansProjects\\SkripsiDavid
       Server\\src\\Server\\hasil.wav");
3   FileOutputStream out = new FileOutputStream(file);
4   out.write(data);
5   out.close();
6   din.close();
7   socket.close();
8   serversocket.close();
9   System.out.println("--Done");
```

BAB 6 PENGUJIAN DAN ANALISIS

Bab ini berisi pengujian dari penelitian implementasi algoritme SPECK pada rekaman audio. Pengujian pada penelitian ini terdiri dari pengujian *test vector*, pengujian fungsional, pengujian validitas enkripsi dan dekripsi, pengujian waktu enkripsi dan dekripsi, dan pengujian keamanan.

6.1 Pengujian Test Vector

Pengujian *test vector* pada SPECK bertujuan untuk membuktikan bahwa *source code* SPECK pada penelitian ini sesuai dengan jurnal "*The Simon And SPECK Families Of Lightweight Block Cipher*" milik Ray Beaulieu. Manualisasi *test vector* telah dijabarkan pada sub bab 4.2.1. Hasil dari program dapat dilihat pada Gambar 6.1.

Plaintext 1 = 6c61766975716520
Plaintext 2 = 7469206564616d20

Kunci 1 = f0e0d0c0b0a0908
Kunci 2 = 706050403020100

Ciphertext 1 = a65d985179783265
Ciphertext 2 = 7860fedf5c570d18

Hasil Dekripsi 1 = 6c61766975716520
Hasil Dekripsi 2 = 7469206564616d20

Gambar 6.1 Hasil Pengujian *Test Vector*

Hasil ini sesuai dengan *test vector* pada jurnal "*The Simon And SPECK Families Of Lightweight Block Cipher*" milik Ray Beaulieu. Hasil test vector pada jurnal ini dapat dilihat pada Tabel 6.1.

Tabel 6.1 Hasil *Test Vector* Algoritme SPECK 128 / 128

SPECK 128 / 128		
Key	0f0e0d0c0b0a0908	0706050403020100
Plaintext	6c61766975716520	7469206564616d20
Ciphertext	a65d985179783265	7860fedf5c570d18

Sumber : (Ray Beaulieu et al, 2013)

Hal ini membuktikan program penulis telah terbukti benar, karena hasil *test vector* penulis sama dengan hasil *test vector* milik Ray Beaulieu.

6.2 Pengujian Fungsional

6.2.1 Pengujian Fungsional *Browse* di Klien

Tabel 6.2 Pengujian Fungsional Fungsi *Browse* di Klien

Nama Kasus Uji	*Browse file* pada klien
Prosedur	- Menekan tombol "Browse" - Mencari *file* rekaman audio - Menekan tombol "Open"
Hasil yang diharapkan	Klien akan menampilkan lokasi *file* rekaman audio yang dipilih
Hasil	Klien menampilkan lokasi *file* rekaman audio yang telah dipilih
Status	Valid

6.2.2 Pengujian Fungsional Masukan *Public Key* Server dan Klien

Tabel 6.3 Pengujian Fungsional Masukan *Public Key* Klien

Nama Kasus Uji	Masukan nilai *public key* klien
Prosedur	- Masukan nilai *public key* klien - Tekan tombol "OK"
Hasil yang diharapkan	Klien akan menampilkan pesan menunggu *public key* server
Hasil	Klien menampilkan pesan menunggu *public key* server
Status	Valid

Tabel 6.4 Pengujian Fungsional Masukan *Public Key* Server

Nama Kasus Uji	Masukan nilai *public key* server
Prosedur	- Masukan nilai *public key* server - Tekan tombol "OK"
Hasil yang diharapkan	Server akan menampilkan hasil *shared key*
Hasil	Server menampilkan hasil *shared key*
Status	Valid

6.2.3 Pengujian Fungsional Bertukar Data *Public Key*

Tabel 6.5 Pengujian Fungsional Bertukar Data *Public Key*

Nama Kasus Uji	Bertukar data *public key*
Prosedur	- Klien mengirimkan *public key* ke server - Server mengirimkan *public key* ke klien
Hasil yang diharapkan	Klien dan server akan menampilkan masing-masing *public key* dan hasil dari *shared key*
Hasil	Klien dan server menampilkan masing-masing *public key* dan hasil dari *shared key*
Status	Valid

6.2.4 Pengujian Fungsional Enkripsi

Tabel 6.6 Pengujian Fungsional Enkripsi Rekaman Audio di Klien

Nama Kasus Uji	Enkripsi rekaman audio di klien
Prosedur	Klien melakukan enkripsi rekaman audio
Hasil yang diharapkan	Klien akan menampilkan pesan sudah selesai enkripsi
Hasil	Klien menampilkan pesan sudah selesai enkripsi
Status	Valid

6.2.5 Pengujian Fungsional Pengiriman *Ciphertext*

Tabel 6.7 Pengujian Fungsional Pengiriman *CIphertext*

Nama Kasus Uji	Pengiriman *ciphertext* dari klien ke server
Prosedur	Klien melakukan pengiriman *ciphertext* ke server
Hasil yang diharapkan	Klien akan menampilkan pesan pengiriman ciphertext telah selesai
Hasil	Klien menampilkan pesan pengiriman ciphertext telah selesai
Status	Valid

6.2.6 Pengujian Fungsional Dekripsi

Tabel 6.8 Pengujian Fungsional Dekripsi *Ciphertext* di server

Nama Kasus Uji	Dekripsi *ciphertext* di server
Prosedur	Server melakukan dekripsi *ciphertext*
Hasil yang diharapkan	Server akan menampilkan pesan dekripsi sudah selesai
Hasil	Server menampilkan pesan dekripsi sudah selesai
Status	Valid

6.2.7 Pengujian Fungsional Menyimpan Rekaman Audio di Server

Tabel 6.9 Pengujian Fungsional Menyimpan Rekaman Audio di Server

Nama Kasus Uji	Menyimpan rekaman audio di server
Prosedur	Server menyimpan rekaman audio di lokasi yang telah dipilih
Hasil yang diharapkan	Server akan menampilkan pesan rekaman audio telah berhasil disimpan
Hasil	Server menampilkan pesan rekaman audio telah berhasil disimpan
Status	Valid

6.3 Pengujian Validitas Enkripsi dan Dekripsi

Pengujian validitas enkripsi dan dekripsi dilakukan untuk mengecek apakah *plaintext* sesuai dengan hasil dekripsi. Penulis menggunakan 7 ekstensi *file* rekaman audio, yaitu AIFF, AMR, FLAC, M4A, MP3, OGG, dan WAV, dengan variasi waktu 5, 30, 60, 120, 180, 240, dan 300 detik. Pengujian validitas akan dijabarkan pada Tabel 6.10 – 6.16.

Tabel 6.10 Pengujian Validitas Pada Ekstensi *File* AIFF

Waktu (detik)	Enkripsi			Dekripsi			Status
5	Plaintext	1	14325900719755694254	Ciphertext	1	17188423110938021216	Valid
		2	13963910676629933517		2	2002633768840270238	
	Ciphertext	1	17188423110938021216	Decrypted	1	14325900719755694254	
		2	2002633768840270238		2	13963910676629933517	
30	Plaintext	1	14325900719760103598	Ciphertext	1	9518250229407295598	Valid
		2	13963910676629933517		2	3752664997090958017	
	Ciphertext	1	9518250229407295598	Decrypted	1	14325900719760103598	
		2	3752664997090958017		2	13963910676629933517	
60	Plaintext	1	14325900719748624558	Ciphertext	1	14875476302727850180	Valid
		2	13963910676629933517		2	399259636878043863	
	Ciphertext	1	14875476302727850180	Decrypted	1	14325900719748624558	
		2	399259636878043863		2	13963910676629933517	
120	Plaintext	1	-	Ciphertext	1	-	-
		2	-		2	-	
	Ciphertext	1	-	Decrypted	1	-	
		2	-		2	-	
180	Plaintext	1	-	Ciphertext	1	-	-
		2	-		2	-	
	Ciphertext	1	-	Decrypted	1	-	
		2	-		2	-	
240	Plaintext	1	-	Ciphertext	1	-	-
		2	-		2	-	
	Ciphertext	1	-	Decrypted	1	-	
		2	-		2	-	
300	Plaintext	1	-	Ciphertext	1	-	-
		2	-		2	-	
	Ciphertext	1	-	Decrypted	1	-	
		2	-		2	-	

Pada Tabel 6.10 menjabarkan tentang pengujian validitas hasil enkripsi dan dekripsi pada ekstensi *file* AIFF. Waktu rekaman audio yang diuji adalah 5, 30, 60, 120, 180, 240, dan 300 detik. Tabel 6.10 dapat dilihat status pada waktu 5, 30, dan 60 detik terlihat valid yang berarti *plaintext* pada enkripsi sama dengan hasil dekripsi. Sedangkan pada 120, 180, 240, dan 300 detik tidak ada hasilnya, melainkan menghasilkan *error java.lang.OutOfMemoryError: Java heap space*.

Hal ini menunjukkan bahwa kebutuhan perangkat keras dan lunak sistem dalam melakukan proses enkripsi dan dekripsi tidak mencukupi kebutuhan sistem.

Tabel 6.11 Pengujian Validitas Pada Ekstensi *File* AMR

Waktu (detik)	Enkripsi			Dekripsi			Status
5	Plaintext	1	11790918389177433141	Ciphertext	1	2129269314412165384	Valid
		2	11930689180897705998		2	2995592997312480264	
	Ciphertext	1	2129269314412165384	Decrypted	1	11790918389177433141	
		2	2995592997312480264		2	11930689180897705998	
30	Plaintext	1	11790918389177433301	Ciphertext	1	487110482462394416	Valid
		2	11640777584224043015		2	9643032755286887949	
	Ciphertext	1	487110482462394416	Decrypted	1	11790918389177433301	
		2	9643032755286887949		2	11640777584224043015	
60	Plaintext	1	11790918389177433141	Ciphertext	1	4538123348508985644	Valid
		2	11928726625925029902		2	7208324510087192941	
	Ciphertext	1	4538123348508985644	Decrypted	1	11790918389177433141	
		2	7208324510087192941		2	11928726625925029902	
120	Plaintext	1	11790918389177433313	Ciphertext	1	14661764342674945641	Valid
		2	11208993933520609294		2	8040452209131061466	
	Ciphertext	1	14661764342674945641	Decrypted	1	11790918389177433313	
		2	8040452209131061466		2	11208993933520609294	
180	Plaintext	1	11790918389177433169	Ciphertext	1	3767724175780430736	Valid
		2	11929009531260059655		2	10137443806401353560	
	Ciphertext	1	3767724175780430736	Decrypted	1	11790918389177433169	
		2	10137443806401353560		2	11929009531260059655	
240	Plaintext	1	11790918389177433295	Ciphertext	1	8699789524468164913	Valid
		2	11785173034485497870		2	9643152234734907971	
	Ciphertext	1	8699789524468164913	Decrypted	1	11790918389177433295	
		2	9643152234734907971		2	11785173034485497870	
300	Plaintext	1	11790918389177433301	Ciphertext	1	18039864077084082346	Valid
		2	10094390073870073870		2	5971450170479307757	
	Ciphertext	1	18039864077084082346	Decrypted	1	11790918389177433301	
		2	5971450170479307757		2	10094390073870073870	

Pada Tabel 6.11 berisi hasil pengujian validitas enkripsi dan dekripsi pada ekstensi *file* AMR. Waktu rekaman audio yang diuji adalah 5, 30, 60, 120, 180, 240, dan 300 detik. Tabel 6.11 dapat dilihat status pada waktu 5, 30, 60, 120, 180, 240, dan 300 detik terlihat valid. Valid ini menandakan bahwa *plaintext* pada enkripsi sama dengan hasil dekripsi.

Tabel 6.12 Pengujian Validitas Pada Ekstensi *File* FLAC

Waktu (detik)	Enkripsi			Dekripsi			Status
5	Plaintext	1	16630915753763176610	Ciphertext	1	14738063340977506530	Valid
		2	10412481220066908032		2	14463417281355904556	
	Ciphertext	1	14738063340977506530	Decrypted	1	16630915753763176610	
		2	14463417281355904556		2	10412481220066908032	
30	Plaintext	1	16630915753763176610	Ciphertext	1	8482161520077531233	Valid
		2	10412481220066980736		2	1635489130605855179	
	Ciphertext	1	8482161520077531233	Decrypted	1	16630915753763176610	
		2	1635489130605855179		2	10412481220066980736	
60	Plaintext	1	16630915753763176610	Ciphertext	1	16011991069678814137	Valid
		2	10412481220067247488		2	2923253564652270462	
	Ciphertext	1	16011991069678814137	Decrypted	1	16630915753763176610	
		2	2923253564652270462		2	10412481220067247488	
120	Plaintext	1	16630915753763176610	Ciphertext	1	567820141944618979	Valid
		2	10412481220067000704		2	5010938044784943884	
	Ciphertext	1	567820141944618979	Decrypted	1	16630915753763176610	
		2	5010938044784943884		2	10412481220067000704	
180	Plaintext	1	16630915753763176610	Ciphertext	1	70010428601442126 76	Valid
		2	10412481220066823808		2	11673274492400564875	
	Ciphertext	1	70010428601442126 76	Decrypted	1	16630915753763176610	
		2	11673274492400564875		2	10412481220066823808	
240	Plaintext	1	16630915753763176610	Ciphertext	1	46605837241721229 56	Valid
		2	10412481220067366272		2	11797288627907794948	
	Ciphertext	1	46605837241721229 56	Decrypted	1	16630915753763176610	
		2	11797288627907794948		2	10412481220067366272	
300	Plaintext	1	16630915753763176610	Ciphertext	1	46754438655998131 36	Valid
		2	10412481220067246976		2	31321229656688618 39	
	Ciphertext	1	46754438655998131 36	Decrypted	1	16630915753763176610	
		2	31321229656688618 39		2	10412481220067246976	

Pada Tabel 6.12 menjabarkan tentang pengujian validitas hasil enkripsi dan dekripsi pada ekstensi *file* FLAC. Pada algoritme SPECK terdapat 2 *word* dengan panjang 64 bit. Tabel 6.12 dapat dilihat status pada waktu 5, 30, 60, 120, 180, 240, dan 300 detik mendapatkan status valid. Valid berarti *plaintext* pada enkripsi sesuai dengan hasil dekripsi.

Tabel 6.13 Pengujian Validitas Pada Ekstensi *File* M4A

Waktu (detik)	Enkripsi			Dekripsi			Status
5	Plaintext	1	9259542228071938544	Ciphertext	1	3466100249967063507	Valid
		2	14822685168677978752		2	13494440425994786961	
	Ciphertext	1	3466100249967063507	Decrypted	1	9259542228071938544	
		2	13494440425994786961		2	14822685168677978752	
30	Plaintext	1	9259542228071938544	Ciphertext	1	3466100249967063507	Valid
		2	14822685168677978752		2	13494440425994786961	
	Ciphertext	1	3466100249967063507	Decrypted	1	9259542228071938544	
		2	13494440425994786961		2	14822685168677978752	
60	Plaintext	1	9259542228071938544	Ciphertext	1	3466100249967063507	Valid
		2	14822685168677978752		2	13494440425994786961	
	Ciphertext	1	3466100249967063507	Decrypted	1	9259542228071938544	
		2	13494440425994786961		2	14822685168677978752	
120	Plaintext	1	9259542228071938544	Ciphertext	1	3466100249967063507	Valid
		2	14822685168677978752		2	13494440425994786961	
	Ciphertext	1	3466100249967063507	Decrypted	1	9259542228071938544	
		2	13494440425994786961		2	14822685168677978752	
180	Plaintext	1	9259542228071938544	Ciphertext	1	3466100249967063507	Valid
		2	14822685168677978752		2	13494440425994786961	
	Ciphertext	1	3466100249967063507	Decrypted	1	9259542228071938544	
		2	13494440425994786961		2	14822685168677978752	
240	Plaintext	1	9259542228071938544	Ciphertext	1	3466100249967063507	Valid
		2	14822685168677978752		2	13494440425994786961	
	Ciphertext	1	3466100249967063507	Decrypted	1	9259542228071938544	
		2	13494440425994786961		2	14822685168677978752	
300	Plaintext	1	9259542228071938544	Ciphertext	1	3466100249967063507	Valid
		2	14822685168677978752		2	13494440425994786961	
	Ciphertext	1	3466100249967063507	Decrypted	1	9259542228071938544	
		2	13494440425994786961		2	14822685168677978752	

Pada Tabel 6.13 berisi tentang hasil pengujian validitas enkripsi dan dekripsi pada ekstensi *file* M4A. Waktu rekaman audio yang diuji adalah 5, 30, 60, 120, 180, 240, dan 300 detik. Pada algoritme SPECK terdapat 2 *word* dengan panjang 64 bit. Tabel 6.13 dapat dilihat status pada waktu 5, 30, 60, 120, 180, 240, dan 300 detik terlihat valid yang menandakan *plaintext* pada enkripsi sama dengan hasil dekripsi.

Tabel 6.14 Pengujian Validitas Pada Ekstensi *File* MP3

Waktu (detik)	Enkripsi			Dekripsi			Status
5	Plaintext	1	9185954438546751616	Ciphertext	1	299043171970181130	Valid
		2	9259542123273814144		2	4538862615545630353	
	Ciphertext	1	299043171970181130	Decrypted	1	9185954438546751616	
		2	4538862615545630353		2	9259542123273814144	
30	Plaintext	1	9185954301107798144	Ciphertext	1	3388462825184529166	Valid
		2	9259542123273814144		2	17136066394315836659	
	Ciphertext	1	3388462825184529166	Decrypted	1	9185954301107798144	
		2	17136066394315836659		2	9259542123273814144	
60	Plaintext	1	9185954301107798144	Ciphertext	1	3388462825184529166	Valid
		2	9259542123273814144		2	17136066394315836659	
	Ciphertext	1	3388462825184529166	Decrypted	1	9185954301107798144	
		2	17136066394315836659		2	9259542123273814144	
120	Plaintext	1	9185954438546751616	Ciphertext	1	6543198140995891572	Valid
		2	9259542123273814144		2	15083657871487042245	
	Ciphertext	1	6543198140995891572	Decrypted	1	9185954438546751616	
		2	15083657871487042245		2	9259542123273814144	
180	Plaintext	1	9185954301107798144	Ciphertext	1	3388462825184529166	Valid
		2	9259542123273814144		2	17136066394315836659	
	Ciphertext	1	3388462825184529166	Decrypted	1	9185954301107798144	
		2	17136066394315836659		2	9259542123273814144	
240	Plaintext	1	9185954438546751616	Ciphertext	1	6543198140995891572	Valid
		2	9259542123273814144		2	15083657871487042245	
	Ciphertext	1	6543198140995891572	Decrypted	1	9185954438546751616	
		2	15083657871487042245		2	9259542123273814144	
300	Plaintext	1	9185954301107798144	Ciphertext	1	3388462825184529166	Valid
		2	9259542123273814144		2	17136066394315836659	
	Ciphertext	1	3388462825184529166	Decrypted	1	9185954301107798144	
		2	17136066394315836659		2	9259542123273814144	

Pada Tabel 6.14 menjabarkan tentang pengujian validitas hasil enkripsi dan dekripsi pada ekstensi *file* MP3. Waktu rekaman audio yang diuji adalah 5, 30, 60, 120, 180, 240, dan 300 detik. Pada algoritme SPECK terdapat 2 word dengan panjang 64 bit. Tabel 6.14 dapat dilihat status pada waktu 5, 30, 60, 120, 180, 240, dan 300 detik terlihat valid. Valid berarti *plaintext* pada enkripsi sama dengan hasil dekripsi.

Tabel 6.15 Pengujian Validitas Pada Ekstensi *File* OGG

Waktu (detik)	Enkripsi			Dekripsi			Status
5	Plaintext	1	14981197581051396224	Ciphertext	1	871445964152078274	Valid
		2	9259542123273788620		2	9401320386339779157	
	Ciphertext	1	871445964152078274	Decrypted	1	14981197581051396224	
		2	9401320386339779157		2	9259542123273788620	
30	Plaintext	1	14981197581051396224	Ciphertext	1	12015316940012109839	Valid
		2	9259542123273800118		2	4024250581609660591	
	Ciphertext	1	12015316940012109839	Decrypted	1	14981197581051396224	
		2	4024250581609660591		2	9259542123273800118	
60	Plaintext	1	14981197581051396224	Ciphertext	1	5890646318733441539	Valid
		2	9259542123273833458		2	11329513940961074212	
	Ciphertext	1	5890646318733441539	Decrypted	1	14981197581051396224	
		2	11329513940961074212		2	9259542123273833458	
120	Plaintext	1	14981197581051396224	Ciphertext	1	68235833998865024 11	Valid
		2	9259542123273786614		2	14987265784598693729	
	Ciphertext	1	68235833998865024 11	Decrypted	1	14981197581051396224	
		2	14987265784598693729		2	9259542123273786614	
180	Plaintext	1	14981197581051396224	Ciphertext	1	12590873318444150441	Valid
		2	9259542123273835002		2	1780819429325960566	
	Ciphertext	1	12590873318444150441	Decrypted	1	14981197581051396224	
		2	1780819429325960566		2	9259542123273835002	
240	Plaintext	1	14981197581051396224	Ciphertext	1	14138446114335354872	Valid
		2	9259542123273793533		2	16680656098033627276	
	Ciphertext	1	14138446114335354872	Decrypted	1	14981197581051396224	
		2	16680656098033627276		2	9259542123273793533	
300	Plaintext	1	14981197581051396224	Ciphertext	1	3455196295606797112	Valid
		2	9259542123273841793		2	1774376677198730105	
	Ciphertext	1	3455196295606797112	Decrypted	1	14981197581051396224	
		2	1774376677198730105		2	9259542123273841793	

Pada Tabel 6.15 berisi hasil pengujian validitas enkripsi dan dekripsi pada ekstensi *file* OGG. Waktu rekaman audio yang diuji adalah 5, 30, 60, 120, 180, 240, dan 300 detik. Tabel 6.15 dapat dilihat status pada waktu 5, 30, 60, 120, 180, 240, dan 300 detik adalah valid. Valid disini menandakan *plaintext* pada enkripsi sama dengan hasil dekripsi.

Tabel 6.16 Pengujian Validitas Pada Ekstensi *File* WAV

Waktu (detik)	Enkripsi			Dekripsi			Status
5	Plaintext	1	15188889775489719168	Ciphertext	1	17011902357171956101	Valid
		2	15546943534130918560		2	13737891781344635183	
	Ciphertext	1	17011902357171956101	Decrypted	1	15188889775489719168	
		2	13737891781344635183		2	15546943534130918560	
30	Plaintext	1	15188889774746816640	Ciphertext	1	218486630096511368	Valid
		2	15546943534130918560		2	10643089124113099825	
	Ciphertext	1	218486630096511368	Decrypted	1	15188889774746816640	
		2	10643089124113099825		2	15546943534130918560	
60	Plaintext	1	15188889774744150400	Ciphertext	1	1818934652812956994	Valid
		2	15546943534130918560		2	7994326146710938454	
	Ciphertext	1	1818934652812956994	Decrypted	1	15188889774744150400	
		2	7994326146710938454		2	15546943534130918560	
120	Plaintext	1	-	Ciphertext	1	-	-
		2	-		2	-	
	Ciphertext	1	-	Decrypted	1	-	
		2	-		2	-	
180	Plaintext	1	-	Ciphertext	1	-	-
		2	-		2	-	
	Ciphertext	1	-	Decrypted	1	-	
		2	-		2	-	
240	Plaintext	1	-	Ciphertext	1	-	-
		2	-		2	-	
	Ciphertext	1	-	Decrypted	1	-	
		2	-		2	-	
300	Plaintext	1	-	Ciphertext	1	-	-
		2	-		2	-	
	Ciphertext	1	-	Decrypted	1	-	
		2	-		2	-	

Pada Tabel 6.16 menjabarkan tentang hasil pengujian validitas enkripsi dan dekripsi pada ekstensi *file* WAV. Pada waktu 5, 30, dan 60 detik terlihat valid yang berarti *plaintext* pada enkripsi sama dengan hasil dekripsi. Sedangkan pada 120, 180, 240, dan 300 detik, program menampilkan *error java.lang.OutOfMemoryError: Java heap space*. Hal ini menunjukkan bahwa kebutuhan perangkat keras dan lunak sistem dalam melakukan proses enkripsi dan dekripsi tidak mencukupi kebutuhan sistem.

6.4 Pengujian Waktu Enkripsi dan Dekripsi

Pengujian waktu enkripsi dan dekripsi dilakukan untuk mengetahui waktu enkripsi dan dekripsi yang diperlukan oleh sistem. Penulis menggunakan 7 ekstensi *file* rekaman audio, yaitu AIFF, AMR, FLAC, M4A, MP3, OGG, dan WAV, dengan variasi waktu 5, 30, 60, 120, 180, 240, dan 300 detik. Pengujian waktu akan dijabarkan pada Tabel 6.17 – 6.23.

Tabel 6.17 Pengujian Waktu Pada Ekstensi *File* AIFF

Waktu (detik)	Ukuran *File* (*bytes*)	Waktu Enkripsi (detik)	Waktu Dekripsi (detik)
5	888.886	19,61	18,77
30	5.298.230	114,68	111,22
60	10.596.406	235,73	230,90
120	21.192.758	-	-
180	31.756.342	-	-
240	42.352.694	-	-
300	52.949.046	-	-

Pada Tabel 6.17 menjabarkan tentang pengujian waktu hasil enkripsi dan dekripsi pada ekstensi *file* AIFF. Waktu rekaman audio yang diuji adalah 5, 30, 60, 120, 180, 240, dan 300 detik. Pada 5, 30, 60 detik terlihat waktu enkripsi lebih besar daripada dekripsi. Selisih waktu enkripsi dan dekripsi pada ekstensi AIFF kurang dari 5 detik. Sedangkan pada 120, 180, 240, dan 300 detik tidak ada hasilnya, melainkan menghasilkan *error java.lang.OutOfMemoryError: Java heap space*. Hal ini menunjukkan bahwa kebutuhan perangkat keras dan lunak sistem dalam melakukan proses enkripsi dan dekripsi tidak mencukupi kebutuhan sistem.

Tabel 6.18 Pengujian Waktu Pada Ekstensi *File* AMR

Waktu (detik)	Ukuran *File* (*bytes*)	Waktu Enkripsi (detik)	Waktu Dekripsi (detik)
5	8.102	0,21	0,26
30	48.102	1,10	1,14
60	96.134	2,11	2,71
120	192.262	4,89	5,24
180	288.070	6,80	6,06
240	384.198	8,98	8,18
300	480.294	10,69	10,19

Pada Tabel 6.18 berisi hasil pengujian waktu enkripsi dan dekripsi pada ekstensi *file* AMR. Waktu rekaman audio yang diuji adalah 5, 30, 60, 120, 180, 240, dan 300 detik. Tabel 6.18 dapat dilihat waktu eksekusi enkripsi pada durasi rekaman 5, 30, 60, dan 120 detik lebih kecil dari dekripsi. Sedangkan pada durasi rekaman 180, 240, dan 300 detik, waktu eksekusi enkripsi lebih besar daripada dekripsi. Selisih waktu enkripsi dan dekripsi pada ekstensi AMR kurang dari 1 detik.

Tabel 6.19 Pengujian Waktu Pada Ekstensi *File* FLAC

Waktu (detik)	Ukuran *File* (*bytes*)	Waktu Enkripsi (detik)	Waktu Dekripsi (detik)
5	201.509	4,42	4,57
30	1.182.363	26,19	24,94
60	2.366.316	51,87	49,79
120	4.731.629	137,38	114,52
180	7.085.676	172,24	179,32
240	9.445.445	210,59	212,03
300	11.813.523	254,44	242,48

Pada Tabel 6.19 menjabarkan tentang pengujian waktu hasil enkripsi dan dekripsi pada ekstensi *file* FLAC. Waktu rekaman audio yang diuji adalah 5, 30, 60, 120, 180, 240, dan 300 detik. Waktu eksekusi untuk enkripsi pada durasi 5, 180, dan 240 detik lebih kecil daripada waktu eksekusi dekripsi. Sedangkan pada durasi 30, 60, 120, dan 300 detik, waktu enkripsi lebih besar daripada dekripsi. Selisih waktu enkripsi dan dekripsi pada ekstensi *file* FLAC kurang dari 23 detik.

Tabel 6.20 Pengujian Waktu Pada Ekstensi *File* M4A

Waktu (detik)	Ukuran *File* (*bytes*)	Waktu Enkripsi (detik)	Waktu Dekripsi (detik)
5	125.138	2,78	2,81
30	741.478	19,72	17,05
60	1.481.670	32,17	36,03
120	2.961.438	82,88	73,52
180	4.436.634	126,11	111,58
240	5.916.402	140,29	135,46
300	7.396.174	160,29	152,72

Pada Tabel 6.20 berisi hasil pengujian waktu enkripsi dan dekripsi pada ekstensi *file* M4A. Tabel 6.20 dapat dilihat waktu eksekusi untuk enkripsi pada 5 dan 60 detik lebih kecil daripada dekripsi. Sedangkan pada durasi 30, 120, 180, 240, dan 300 detik, waktu untuk melakukan enkripsi lebih besar daripada dekripsi. Selisih waktu eksekusi enkripsi dan dekripsi pada ekstensi M4A kurang dari 15 detik.

Tabel 6.21 Pengujian Waktu Pada Ekstensi *File* MP3

Waktu (detik)	Ukuran *File* (*bytes*)	Waktu Enkripsi (detik)	Waktu Dekripsi (detik)
5	68.820	1,59	1,60
30	414.740	9,38	9,06
60	827.297	22,75	20,38
120	1.653.052	47,38	43,93
180	2.473.348	64,89	57,40
240	3.302.004	82,71	81,64
300	4.127.562	99,32	97,87

Pada Tabel 6.21 menjabarkan tentang hasil pengujian waktu enkripsi dan dekripsi pada ekstensi *file* MP3. Waktu enkripsi pada durasi rekaman 5 detik lebih kecil dibandingkan dengan waktu eksekusi dekripsi. Sedangkan pada durasi

rekaman 30, 60, 120, 180, 240, dan 300 detik, waktu enkripsi lebih besar daripada dekripsi. Pada Tabel 6.21 dapat dilihat selisih waktu eksekusi enkripsi dan dekripsi pada ekstensi MP3 kurang dari 7 detik.

Tabel 6.22 Pengujian Waktu Pada Ekstensi *File* OGG

Waktu (detik)	Ukuran *File* (*bytes*)	Waktu Enkripsi (detik)	Waktu Dekripsi (detik)
5	79.360	1,76	1,86
30	454.115	9,97	9,77
60	903.810	23,33	22,87
120	1.803.200	51,74	46,80
180	2.699.830	66,63	60,78
240	3.599.220	79,04	76,47
300	4.498.610	98,61	93,47

Pada Tabel 6.22 berisi pengujian waktu enkripsi dan dekripsi pada ekstensi *file* OGG. Waktu eksekusi enkripsi pada 5 detik lebih kecil daripada dekripsi. Sedangkan pada durasi rekaman audio sebesar 30, 60, 120, 180, 240, dan 300 detik, enkripsi membutuhkan waktu lebih lama daripada dekripsi. Selisih waktu enkripsi dan dekripsi pada ekstensi OGG kurang dari 6 detik.

Tabel 6.23 Pengujian Waktu Pada Ekstensi *File* WAV

Waktu (detik)	Ukuran *File* (*bytes*)	Waktu Enkripsi (detik)	Waktu Dekripsi (detik)
5	1.777.752	37,70	36,89
30	5.298.220	112,57	108,20
60	10.596.396	233,05	231,10
120	21.192.748	-	-
180	31.756.332	-	-
240	42.352.684	-	-
300	52.949.036	-	-

Pada Tabel 6.23 menjabarkan tentang hasil pengujian waktu enkripsi dan dekripsi pada ekstensi *file* WAV. Waktu enkripsi pada durasi rekaman 5, 30, dan 60 detik lebih besar daripada dekripsi. Selisih waktu enkripsi dan dekripsi pada ekstensi WAV kurang dari 4 detik. Sedangkan pada 120, 180, 240, dan 300 detik tidak ada hasilnya, melainkan menghasilkan *error java.lang.OutOfMemoryError: Java heap space*. Hal ini menunjukkan bahwa kebutuhan perangkat keras dan lunak sistem dalam melakukan proses enkripsi dan dekripsi tidak mencukupi kebutuhan sistem.

Pada Tabel 6.17–Tabel 6.23 dapat dianalisa bahwa waktu enkripsi dan dekripsi dipengaruhi oleh ukuran *file* rekaman audio. Semakin besar ukuran *file* rekaman audio, maka semakin besar juga waktu untuk melakukan enkripsi dan dekripsi. Selisih waktu enkripsi dan dekripsi kurang dari 23 detik. Kemudian ada beberapa *file* tidak bisa dienkripsi karena ukuran *file* yang cukup besar. Sistem ini hanya dapat berjalan pada besar *file* maksimum 20 MB. Hal ini dikarenakan keterbatasan kebutuhan perangkat keras dan lunak sistem.

6.5 Pengujian Waktu Sistem

Pengujian waktu sistem dilakukan untuk mengetahui dan membandingkan waktu eksekusi sistem dengan variasi ekstensi *file* rekaman audio. Penulis akan menggunakan rekaman audio berdurasi 60 detik dengan ekstensi *file* AIFF, AMR, FLAC, M4A, MP3, OGG dan WAV. Masing-masing ekstensi *file* akan dilakukan pengujian sebanyak 30 kali. Ukuran *file* rekaman audio yang berdurasi 60 detik pada masing-masing ekstensi file dapat dilihat pada Tabel 6.24.

Tabel 6.24 Ukuran *File* Rekaman Audio berdurasi 60 detik

Ekstensi *File*	Ukuran *File (bytes)*
AIFF	10.596.406
AMR	96.134
FLAC	2.366.316
M4A	1.481.670
MP3	827.297
OGG	903.810
WAV	10.596.396

Hasil pengujian sistem dengan variasi ekstensi *file* dapat dilihat pada Tabel 6.25.

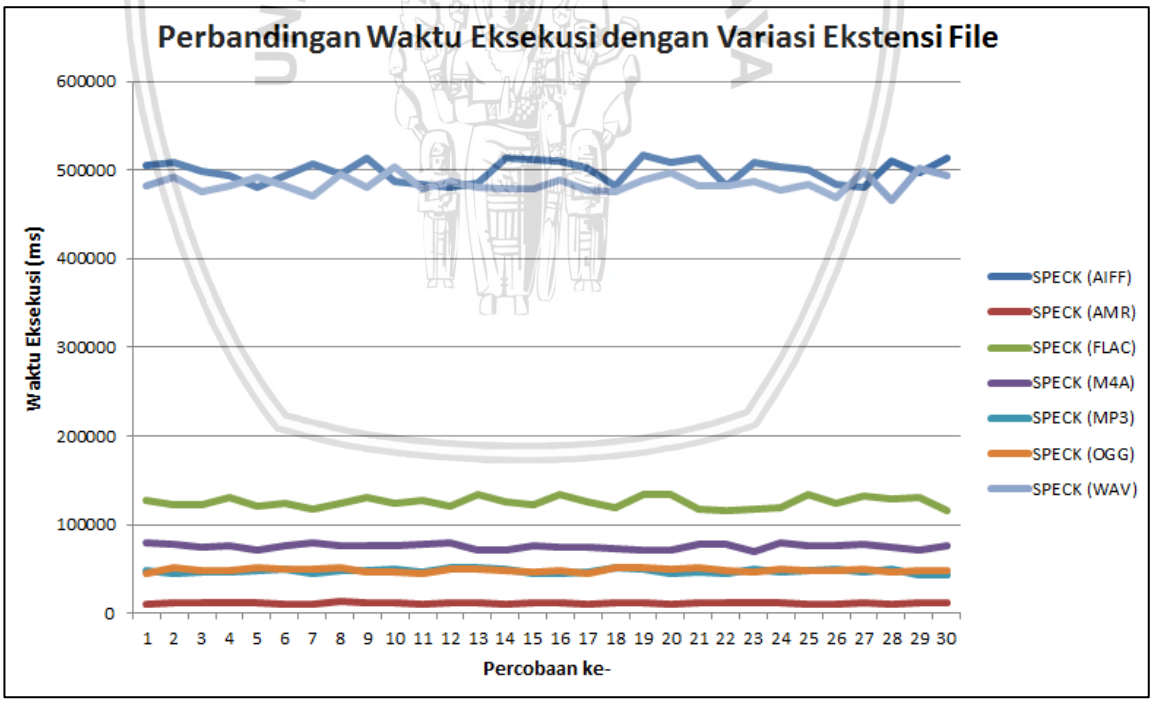
Tabel 6.25 Pengujian Waktu Sistem

Percobaan	AIFF (detik)	AMR (detik)	FLAC (detik)	M4A (detik)	MP3 (detik)	OGG (detik)	WAV (detik)
1	504,94	10,49	127,97	79,02	48,69	45,03	481,66
2	508,66	12,10	122,42	78,15	44,10	51,68	492,71
3	498,32	11,65	122,01	74,08	46,19	47,74	475,22
4	494,31	11,94	130,13	76,79	46,63	48,63	482,03
5	480,58	11,12	121,46	71,12	48,26	50,67	491,93
6	493,03	10,73	123,96	75,99	49,59	50,44	482,48
7	506,40	10,51	118,23	79,64	44,73	49,30	470,75
8	496,18	12,72	123,97	75,70	47,67	51,07	495,42
9	513,08	11,30	131,05	76,13	48,48	46,46	480,58
10	487,11	11,74	124,56	75,83	50,59	47,31	503,61
11	483,05	10,93	126,71	78,36	47,29	45,17	478,20
12	480,93	11,56	120,36	79,46	50,76	49,63	487,45
13	485,35	11,86	133,41	71,99	51,26	50,36	481,15
14	513,79	10,56	125,41	71,78	50,29	48,06	479,14
15	511,77	12,03	122,12	76,08	45,06	46,68	479,14
16	510,64	11,51	133,48	74,37	45,23	47,54	488,50
17	501,39	10,85	125,41	75,07	46,84	45,25	476,54
18	481,85	11,20	119,16	73,31	51,59	50,82	475,52
19	516,81	11,46	133,93	70,84	49,41	51,29	489,13
20	508,52	10,50	134,18	72,07	45,27	49,58	496,30

Tabel 6.26 Pengujian Waktu Sistem (Lanjutan)

Percobaan	AIFF (detik)	AMR (detik)	FLAC (detik)	M4A (detik)	MP3 (detik)	OGG (detik)	WAV (detik)
21	513,96	11,80	116,93	77,11	47,10	50,71	482,30
22	482,45	11,07	115,90	77,72	44,74	47,99	482,79
23	509,18	11,11	117,75	70,29	49,73	46,19	487,85
24	503,22	11,68	118,94	79,33	46,53	49,98	477,19
25	501,09	10,45	133,40	76,99	48,12	48,17	484,65
26	484,20	10,55	124,81	75,46	50,59	47,63	468,89
27	481,05	11,43	131,53	78,28	46,40	50,28	498,31
28	509,74	10,79	129,47	73,78	49,75	46,44	465,74
29	497,49	11,85	130,31	71,26	44,02	47,93	502,58
30	513,60	11,19	115,59	76,55	43,88	48,04	494,52
Rata-rata	499,09	11,29	125,15	75,42	47,63	48,53	484,41

Tabel 6.25 menampilkan perbandingan waktu eksekusi dengan masukan rekaman audio berdurasi 60 detik dan variasi ekstensi *file*. Pengujian ini dilakukan pada masing-masing ekstensi *file* sebanyak 30 kali. Hasil dari Tabel 6.25 dibuat grafik yang dapat dilihat pada Gambar 6.2.



Gambar 6.2 Grafik Perbandingan Waktu Eksekusi dengan Variasi Ekstensi *File*

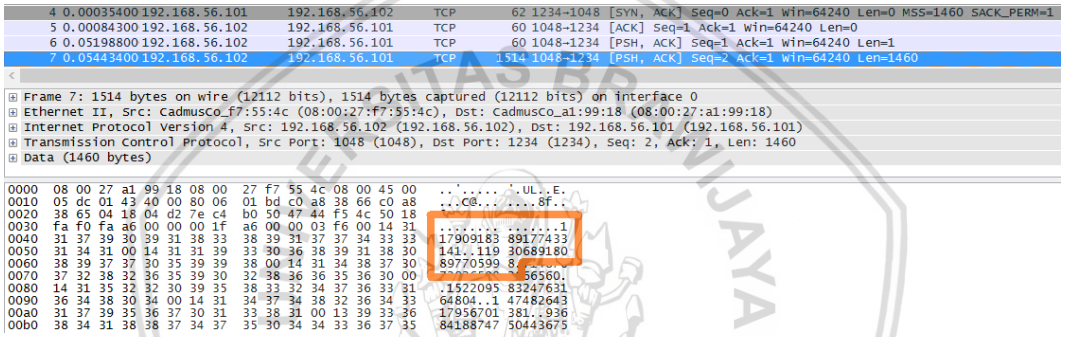
Pada Tabel 6.24, Tabel 6.25, dan Gambar 6.2, dapat dilihat bahwa ekstensi *file* AIFF memiliki rata-rata waktu eksekusi paling lama. Sedangkan pada ekstensi *file* AMR memiliki rata-rata waktu eksekusi paling kecil. Hal ini disebabkan karena adanya perbedaan ukuran *file* pada durasi rekaman audio yang sama. Perbedaan ukuran *file* pada variasi ekstensi *file* dapat dilihat pada Tabel 6.24. Dari analisis ini

dapat disimpulkan bahwa semakin besar ukuran *file* rekaman audio yang ingin dikirim oleh klien, maka semakin besar waktu eksekusi yang diperlukan oleh sistem.

6.6 Pengujian Keamanan

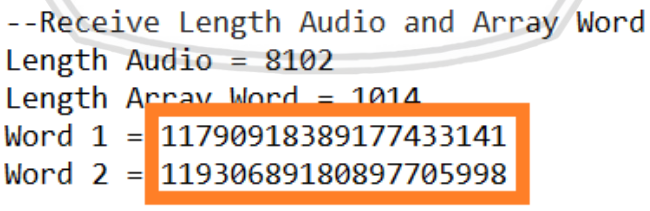
Pengujian keamanan dilakukan dengan cara menangkap komunikasi server dan klien dengan menggunakan program wireshark. Pengujian ini dilakukan pada sistem yang menggunakan algoritme SPECK dan tanpa algoritme SPECK.

Pada sistem tanpa menggunakan algoritme SPECK, klien akan mengirimkan *array word* ke server. *Word* berisi bilangan 64 bit. Hasil *sniffing* dengan wireshark dan hasil output program sistem tanpa algoritme SPECK dapat dilihat pada Gambar 6.3 dan Gambar 6.4.



Gambar 6.3 Hasil *Sniffing* pada Sistem tanpa Algoritme SPECK

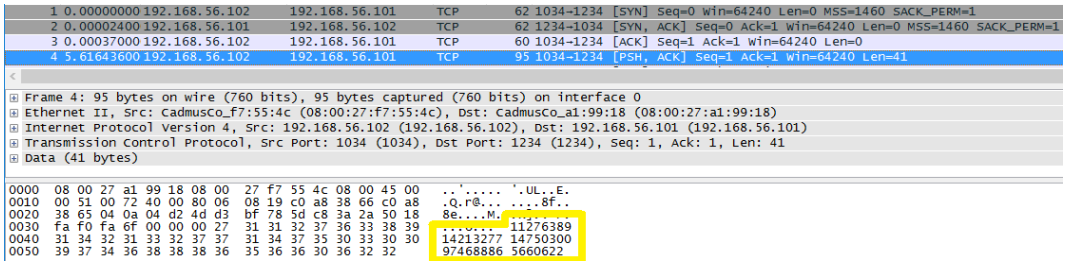
Gambar 6.3 menunjukkan klien dengan *IP address* 192.168.56.102 (*port* 1048) mengirimkan data kepada server dengan *IP address* 192.168.56.101 (*port* 1234). Komunikasi ini menggunakan protokol TCP. Klien mengirimkan data berupa *word* 1 dan 2. *Word* 1 dan 2 dapat dilihat pada kotak pada Gambar 6.3.



Gambar 6.4 Hasil Output Program pada Sistem tanpa Algoritme SPECK

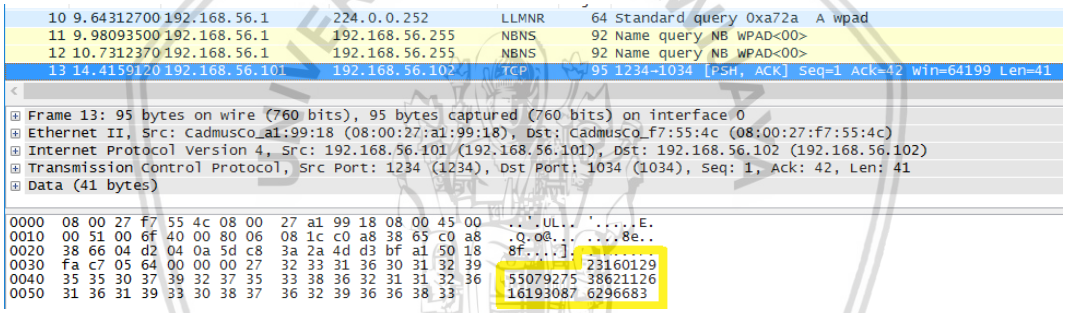
Gambar 6.3 menunjukkan *word* 1 dan 2 pada hasil *sniffing* sesuai dengan hasil output program pada Gambar 6.4. Bila semua *word* ini digabungkan dan diubah ke rekaman audio, maka orang ketiga dapat mendengarkan rekaman audio tersebut. Hal ini membuat rekaman audio tersebut tidak bersifat rahasia.

Pada sistem yang menggunakan algoritme SPECK, pertama klien akan mengirimkan *public key*-nya ke server. Kemudian, server akan mengirimkan *public key*-nya ke klien. Terakhir, klien akan mengirimkan *ciphertext* ke server. Hasil *sniffing* dengan wireshark dan hasil output program sistem dengan algoritme SPECK dapat dilihat pada Gambar 6.5-Gambar 6.9.



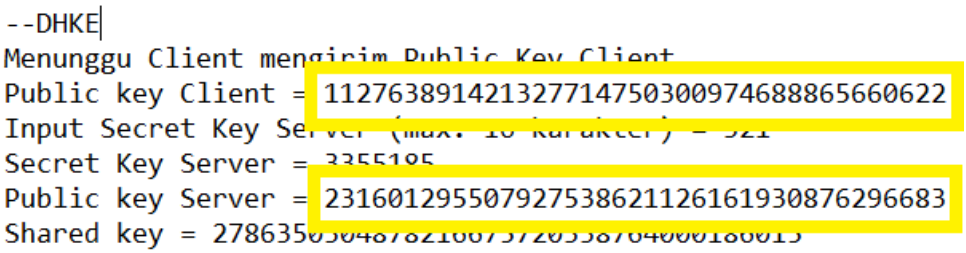
Gambar 6.5 Hasil *Sniffing Public Key* Klien pada Sistem dengan Algoritme SPECK

Gambar 6.5 menunjukkan klien dengan *IP address* 192.168.56.102 (*port* 1034) mengirimkan data kepada server dengan *IP address* 192.168.56.101 (*port* 1234). Komunikasi ini menggunakan protokol TCP. Klien mengirimkan *public key*-nya ke server. *Public key* klien dapat dilihat pada kotak pada Gambar 6.5.



Gambar 6.6 Hasil *Sniffing Public Key* Server pada Sistem dengan Algoritme SPECK

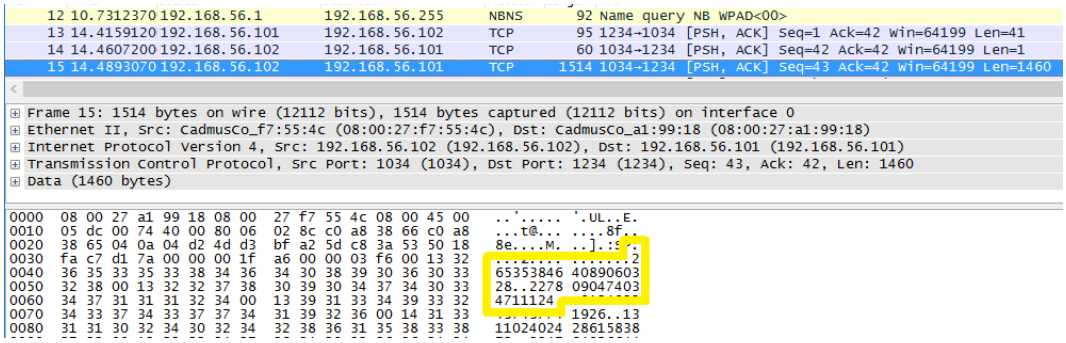
Gambar 6.6 menunjukkan server dengan *IP address* 192.168.56.101 (*port* 1234) mengirimkan data kepada klien dengan *IP address* 192.168.56.102 (*port* 1034). Komunikasi ini menggunakan protokol TCP. Server mengirimkan *public key*-nya ke klien. *Public key* server dapat dilihat pada kotak pada Gambar 6.6.



Gambar 6.7 Hasil Output Program *Public Key* Klien dan Server pada Sistem dengan Algoritme SPECK

Gambar 6.5 dan Gambar 6.6 menunjukkan *public key* klien dan server pada hasil *sniffing* sesuai dengan hasil output program pada Gambar 6.7. Hal ini *public*

key telah dikirim dengan benar. *Public key* ini akan diproses menjadi *shared key* yang akan menjadi kunci untuk melakukan enkripsi dan dekripsi.



Gambar 6.8 Hasil *Sniffing Ciphertext* pada Sistem dengan Algoritme SPECK

Gambar 6.8 menunjukkan klien dengan *IP address* 192.168.56.102 (*port* 1034) mengirimkan data kepada server dengan *IP address* 192.168.56.101 (*port* 1234). Komunikasi ini menggunakan protokol TCP. Klien mengirimkan *ciphertext* 1 dan 2 ke server. *Ciphertext* 1 dan 2 dapat dilihat pada kotak pada Gambar 6.8.

```
--Decrypt
Ciphertext 1 = 26535384640890603 28
Ciphertext 2 = 22780904740347111 24
Decrypted 1 = 11790918389177433141
Decrypted 2 = 11930689180897705998
```

Gambar 6.9 Hasil Output Program *Ciphertext* pada Sistem dengan Algoritme SPECK

Gambar 6.8 menunjukkan *ciphertext* 1 dan 2 pada hasil *sniffing* sesuai dengan hasil output program pada Gambar 6.9. Bila semua *word* ini digabungkan dan diubah ke rekaman audio, maka orang ketiga tidak dapat mendengarkan rekaman audio tersebut. Hal ini disebabkan klien telah melakukan enkripsi dengan algoritme SPECK sebelum dikirimkan ke server. Untuk mendengarkan rekaman audio tersebut, orang ketiga perlu melakukan dekripsi dengan algoritme SPECK. Oleh karena itu, rekaman audio akan bersifat aspek kerahasiaan.

BAB 7 PENUTUP

Bab ini berisi kesimpulan dan saran penelitian implementasi algoritme SPECK dalam mengamankan pengiriman rekaman audio.

7.1 Kesimpulan

Berdasarkan hasil pengujian yang telah dilakukan dapat disimpulkan, sebagai berikut :

1. Algoritme SPECK dapat diterapkan pada proses pengiriman rekaman audio. Algoritme SPECK akan memberikan aspek kerahasiaan kepada pengguna. Hal ini dibuktikan pada pengujian keamanan. Algoritme SPECK akan menghasilkan *ciphertext* yang sangat berbeda dengan *plaintext*. Bila *ciphertext* ini digabungkan dan diubah menjadi rekaman audio, maka pengguna tidak dapat mendengarkan rekaman audio tersebut. Agar pengguna dapat mendengarkan rekaman audio tersebut, diperlukan dekripsi dengan algoritme SPECK sebelum diubah menjadi rekaman audio.
2. Dengan menerapkan algoritme SPECK, rata-rata waktu eksekusi yang diperlukan untuk melakukan enkripsi lebih besar daripada dekripsi. Selisih waktu eksekusi enkripsi dan dekripsi kurang dari 23 detik.
3. Algoritme SPECK dapat melakukan enkripsi dan dekripsi pada beberapa ekstensi *file*, yaitu AIFF, AMR, FLAC, M4A, MP3, OGG, dan WAV. Ekstensi *file* tidak mempengaruhi pada lama waktu eksekusi melakukan enkripsi. Hal yang mempengaruhi waktu eksekusi enkripsi dan dekripsi adalah ukuran *file* rekaman audio. Rata-rata waktu eksekusi sistem yang paling besar adalah AIFF sebesar 499,09 detik. Sedangkan rata-rata waktu eksekusi sistem yang paling kecil adalah AMR sebesar 11,29 detik.Pada penelitian ini, sistem hanya dapat melakukan enkripsi dan dekripsi pada maksimum ukuran *file* sebesar 20 MB karena keterbatasan perangkat lunak dan perangkat keras.

7.2 Saran

Saran penulis untuk penelitian berikutnya adalah melakukan analisis perbandingan waktu tempuh dan tingkat keamanan dengan algoritme *stream cipher* atau *block cipher* yang lain. Selain itu, penelitian ini dapat dikembangkan dari sisi tampilan agar dapat digunakan oleh masyarakat. Diharapkan penelitian ini dapat digunakan sebagai rujukan untuk penelitian selanjutnya.

DAFTAR PUSTAKA

Adhika, A. A. Ngurah Pradnya, 2012. *Enkripsi dan Dekripsi Audio Format AMR dengan Algoritma Kriptografi LOKI97*. S1. Universitas Udayana.

Ariyus, D., 2008. *Pengantar Ilmu Kriptografi : Teori, Analisis, dan Implementasi*. Yogyakarta : Andi Offset.

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B. & Wingers, L., 2013. *The Simon and Speck Families of Lightweight Block Ciphers*. USA.

Fidens, F., 2006. *Dasar Kriptografi*. [Online] Available at: http://www.ilmukomputer.com [Accessed 3 Maret 2018].

Jawahir, A. & Haviluddin, 2015. *An Audio Encryption using transposition method*. Universitas Mulawarman.

Kurniawan, Y., 2004. *Keamanan Internet dan Jaringan Telekomunikasi*. Bandung: Informatika Bandung.

Purwosasongko, D. T., 2015. *Kekuatan Alat Bukti Rekaman Suara dalam Proses Pemberantasan Tindak Pidana Korupsi*. S1. Universitas Muhammadiyah Surakarta.

Rahardjo, B., 2005. *Keamanan Sistem Informasi Berbasis Internet*. Bandung: PT. Insan Indonesia.

Schildt, H., 2007. *Java : The Complete Reference*. New York: McGraw-Hill Education.

Schneier, B., 1996. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 2nd ed.

Sentot, K., 2009. *Teori & Aplikasi Kriptografi*. SPK IT Consulting.

Sitompul, Josua, 2012. *Cyberspace, Cybercrimes, Cyberlaw: Tinjauan Aspek Hukum Pidana*. Jakarta: Tatanusa.

Stalling, W., 2005. *Cryptography and Network Security Principals and Practice, Fourth Edition*. S.l. : Prentice Hall.